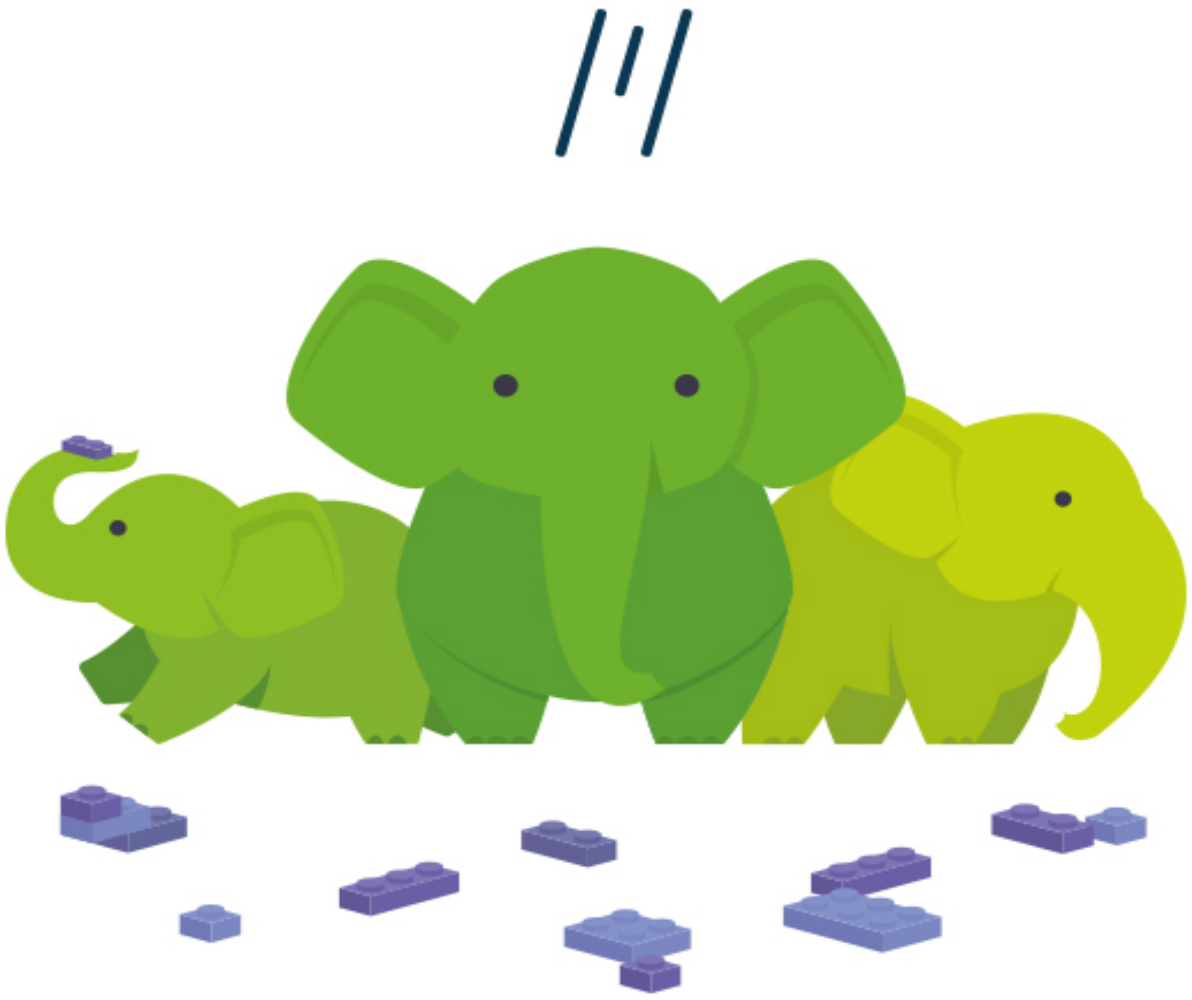
Do PHP ao Laminas

Domine as boas práticas



Casa do Código

FLÁVIO LISBOA

Sumário

- ISBN
- Agradecimentos
- Sobre o autor
- Prefácio
- 1. Introdução
- 2. Boas práticas de desenvolvimento
- 3. Bússola do ambiente de desenvolvimento
- 4. Bússola da estrutura de PHP
- 5. Bússola de funções e classes de PHP
- 6. MVC e MVVM com Laminas
- 7. Mapeamento objeto-relacional
- 8. Formulários dinâmicos
- 9. Visão e controle com relacionamentos
- 10. Guia de referência rápida do MVC do Laminas
- 11. Considerações finais
- 12. Referencial teórico
- 13. Referências

ISBN

Impresso e PDF: 978-65-86110-49-4

EPUB: 978-85-94188-52-6

MOBI: 978-85-94188-53-3

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

Agradecimentos

Agradeço a Deus, pelos milagres e por me sustentar na dificuldade.

Agradeço à minha esposa, por ser minha companheira.

Agradeço à minha filha, por ser mais do que eu sonhava quando a imaginei.

Agradeço a toda a equipe da Casa do Código, por tornar o projeto deste livro realidade, principalmente à Vivian, que transformou o caos do meu pensamento em algo legível.

Agradeço aos meus colegas da comunidade PHP, que me ensinaram muito e mostraram caminhos para trabalhar melhor. E a todos os leitores que enviaram perguntas, críticas, sugestões e correções.

Agradeço aos alunos que treinei nos últimos dez anos, que compartilharam seus anseios e realidades, e ampliaram meu horizonte.

Agradeço aos artistas de histórias em quadrinhos, por estimularem minha imaginação e mostrarem que é possível aprender com diversão.

Agradeço aos excelentes professores que tive a honra de conhecer.

Agradeço à minha mãe, onde quer que ela esteja, e aos que duvidaram do que eu era capaz de fazer.

Sobre o autor

Flávio Gomes da Silva Lisboa é mestre em Tecnologia e Sociedade pela Universidade Tecnológica Federal do Paraná e bacharel em Ciência da Computação, com especialização em Tecnologia Java. Atualmente cursa o doutorado em Tecnologia e Sociedade.

Programador formado pelo Centro Estadual de Educação Tecnológica Paula Souza, já atuou em empresas privadas de TI e foi funcionário do Banco do Brasil, onde atuou como analista na diretoria internacional.

É analista de desenvolvimento do Serviço Federal de Processamento de Dados (Serpro), no qual foi coordenador do *Programa Serpro de Software Livre* e gerente de equipe de desenvolvimento.

Tem mais de 10 anos de experiência em treinamento de desenvolvedores em programação orientada a objetos, padrões de projeto e uso de frameworks.

Lecinou na pós-graduação da UNICID, em São Paulo, e das faculdades Sant'Anna, em Ponta Grossa. Atualmente é professor de pós-graduação na UNICESUMAR, em Maringá, e na Faculdade Alfa de Umuarama e de graduação no Isulpar, em Paranaguá.

É pioneiro na bibliografia em língua portuguesa sobre Zend Framework e Symfony. Por fim, ele é associado a *ABRAPHP*, *Zend PHP Certified Engineer*, *Zend Framework Certified Engineer* e *Zend Framework 2 Certified Architect*.

É autor ainda do documentário *Rom, Biografia não Autorizada*, e da ficção *O UM: a solidão e a harmonia*.

Prefácio

De acordo com Parish (1990, p. 7), *“temos de lembrar que quase todo software passa por manutenção de um tipo ou de outro durante a sua vida”.* Por isso, é bom ter uma ferramenta para nos ajudar a criar programas fáceis de manter. É o que Laminas faz para a linguagem de programação PHP, usando o paradigma da Orientação a Objetos.

Este é um livro para quem quer aprender Laminas de verdade. Não é uma *apostilinha* com uma foto bacana na capa, uma mera tradução da documentação, ou uma enorme lista de tópicos iguais a qualquer tutorial da internet – que termina antes mesmo de você ter suas dúvidas esclarecidas.

Se você quer realmente aprender a programar em PHP direito e quer desenvolver com componentes reutilizáveis que podem ser adaptados às suas necessidades – pensando no melhor compromisso entre controle e desempenho –, este é o livro certo.

Você pode utilizar o Laminas (apelido que demos e será usado daqui em diante) como *container* da sua aplicação, dando todo o poder a ele; ou apenas usar dos componentes que lhe forem convenientes. Este livro tem como foco a primeira opção.

Ele foi escrito para quem programa em PHP, tendo em mente que telepatia não existe e nada é óbvio – a não ser que esteja explícito. Não é um livro para fazer você dormir, mas pode eventualmente evitar que você passe a noite em claro.

O conteúdo foi organizado da seguinte forma: no capítulo 1, fazemos a introdução à nossa emocionante aventura. Primeiro, preparamos o terreno fértil de sua mente, ao desmistificar os termos da arquitetura de software, até chegarmos ao conceito de framework. Isso porque usar algo sem compreender bem do que se trata é como dar o anel do Lanterna Verde ao Patolino.

No capítulo 2, apresentaremos vários padrões e recomendações de desenvolvimento gerais e orientados para PHP. Já no capítulo 3, vamos instalar as ferramentas necessárias e configurar nosso ambiente de trabalho.

Os capítulos 4 e 5 são os guias de referência PHP para este livro. Seu primeiro impulso certamente será pulá-los, mas você poderá retornar sempre que precisar de um fundamento de PHP. Assim, nos capítulos 6 a 9, aplicamos o padrão MVC com Laminas, aprendendo a dar o domínio da nossa aplicação ao framework. Vamos confiar a ele o controle sobre o nosso código. Esses capítulos abordam os componentes *Laminas\Mvc* e *Laminas\View*, e o componente de geração de formulários dinâmicos, o *Laminas\Form*.

No capítulo 10, apresentamos um guia de referência rápida para a implementação MVC do Laminas, que será o seu auxiliar imediato para dúvidas sobre o que você implementou nos capítulos anteriores e para os novos projetos que usarem o Laminas. No capítulo 11, encerraremos o livro com algumas considerações. Por fim, vamos apresentar nosso referencial teórico no capítulo 12, caso você queira refletir sobre o que leu.

Fique tranquilo(a) com relação ao software necessário para fazer bom proveito deste livro, pois todas as ferramentas que usaremos para desenvolvimento do Laminas são livres, como o próprio framework. Este livro foi escrito com base na crença de que devemos remover – e não criar – obstáculos para a aquisição de conhecimento. Restringir o acesso ao conhecimento é uma grande bobagem, e o único efeito concreto que causa é a escassez de bons profissionais.

No final do século XIX, o Japão pulou uma era inteira para alcançar o nível de desenvolvimento tecnológico do Ocidente e tornar-se uma superpotência. Em vez de sucumbirem ao próprio orgulho, tentando fazer as coisas por conta própria, os líderes e técnicos japoneses foram sensatos o bastante para conhecer o que outros países

haviam feito e simplesmente assimilarem esse conhecimento. Como veremos durante a leitura, ter é melhor que ser, mas usar é melhor que ter.

Decerto que temos capacidade para criar softwares nacionais (incluindo frameworks), entretanto, não podemos nos esquecer de que disputamos a corrida em condições desiguais. Logo, a estratégia Meiji pode ser a mais adequada em certas situações. Também temos de respeitar a "experiência dos ancestrais".

Assim é a ciência. Nada se faz do zero. Parafraseando Isaac Newton: não podemos fazer nada grandioso se não nos apoiarmos sobre ombros de gigantes. Então, aproprie-se do poder do Laminas e explore seu potencial para criar software de qualidade.

Todo código-fonte dos projetos deste livro está disponível no GitHub, em https://github.com/fgsl/zf3booksamples. Saberei se você leu o prefácio se enviar um e-mail perguntando sobre isso. ;)

Boa sorte, e que o PHP esteja com você!

Capítulo 1
Introdução

Os sistemas de computadores, na grande maioria, são difíceis de alterar. Eles assumem o que você precisa antes mesmo de começar a construí-los. – John J. Patrick

Laminas é um framework de código aberto para o desenvolvimento de aplicações e serviços Web com PHP. De acordo com as premissas de um framework de software, ele é implementado usando programação orientada a objetos, o que implica que veremos um pouco sobre isto neste livro.

Outra característica fundamental é que, desde sua primeira encarnação como Zend Framework, o Laminas segue uma filosofia de componentes *use quando quiser*. Mas quais as razões para usarmos Orientação a Objetos e componentes na construção de software?

Ada Lovelace faz uma definição de software de uma beleza tal que faz jus a seu pai, o poeta Lord Byron. Em poucas palavras, ela afirma que um software pode fazer com que um computador seja capaz de fazer qualquer coisa. Um computador é uma máquina de propósito geral. Sozinho, ele não faz nada. Porém, ele pode ser o núcleo de um sistema complexo. Ele pode controlar lavadoras, fornos, carros, enceradeiras e rádios.

Mas o que a princípio pode parecer uma panaceia pode revelar-se uma grande armadilha. Isso porque, embora seja imaterial, o software precisa de manutenção. A manutenção é um problema recorrente para quem desenvolve software, e um dos objetivos deste livro é ajudá-lo a resolver esse problema – ou ao menos minimizá-lo.

1.1 Manutenção de software

O custo é um fator crucial na escolha por um produto. Mas às vezes consideramos apenas o custo de aquisição do bem, sendo que há um outro, proporcional ao seu tempo de consumo, que é o custo de manutenção. Software é um produto, ele não deixa de existir após o utilizarmos. E para que ele continue funcionando de forma satisfatória, é necessário realizar ajustes, mudanças e reparos.

Precisamos encarar o software da mesma forma como encaramos um carro ou um móvel de nossa casa. O carro precisa de revisões periódicas, pois as peças sofrem desgaste. O móvel, como seu nome sugere, pode mudar de lugar, e para isso deve ser fácil movê-lo, desmontá-lo e remontá-lo. Se for um móvel embutido, ele não pode criar impedimentos para reparos, como a vedação de uma tomada ou do orifício por onde passa o cabo da antena.

A motocicleta do desenho *Carangos e Motocas* costumava dizer: “eu te disse, eu te disse”. Bem, alguns autores disseram que o software já era complicado há uma década e ficaria ainda mais. Hoje, eles bem poderiam dizer: "nós dissemos, nós dissemos".

Manter software é algo similar a ser protagonista de um filme de kung-fu produzido em Hong Kong nos anos 1970: você luta sozinho com 30 caras ao mesmo tempo.

O motivo é simples: a partir de um momento, nós perdemos o controle sobre o software. Perder o controle sobre o software não quer dizer que não sabemos mais o que ele faz, mas, sim, que não sabemos mais como ele faz. Nós deixamos de dominar o software e passamos a operá-lo, como um operário alienado que manuseia uma máquina sem saber exatamente como ela funciona.

Temos pessoas criando software complexo e precisamos que ele seja fácil de ser mantido. É o paradoxo da programação.

Mais código, mais problemas

Bem, você percebeu que tem problemas, mas isso certamente você já sabia, só não queria aceitar. Porém, o primeiro passo para resolver problemas é a aceitação, reconhecer que os problemas de manutenção existem. O segundo passo é mensurá-los de forma realista, porque, imediatamente após reconhecê-los, você vai subestimá-los, dizendo “ah, não é tão grave assim”. Só que os problemas de manutenção de software são mais graves do que você pensa, e eles só vão piorar com o tempo.

O programador não é comerciante de frutas e verduras, mas tem de lidar com abacaxis e pepinos com grande frequência. Não se pode evitar que eles apareçam, mas podemos trabalhar para minimizar sua ocorrência e diminuir seu impacto. Embora ninguém queira reconhecer isso – pois não é comercialmente favorável –, problemas não se resolvem completamente na realidade. Eles são reduzidos até uma condição em que seus efeitos sejam irrelevantes.

Você pode gastar muito tempo para compreender o que o cliente ou o usuário deseja, mas o fato é que a única coisa que certamente vai aumentar entre o início e o fim do projeto é a quantidade de código-fonte. Código-fonte é como praga, erva daninha e coelhos na Austrália: a tendência é só aumentar e se espalhar.

Desenvolver software implica gerar código-fonte. Quanto mais código-fonte tem um software, mais complexo ele é. Quanto mais complexo, mais difícil fica mantê-lo. Quanto mais difícil é manter algo funcionando, mais caro isso fica. Ou seja, se um software é difícil de manter, isso implica que qualquer mudança é demorada.

Seria desnecessário dizer isso, mas temos de ressaltar que demorar significa que levará um tempo para ser feito. Tempo é dinheiro, como diria o Tio Patinhas, e, se você não tem tempo, logo, você não tem dinheiro. De uma forma mais clara, se você não tem tempo para fazer manutenção, a qualidade do software cairá; ele vai degradar e gerar mais problemas, e você perderá dinheiro (ou simplesmente não conseguirá ganhá-lo).

A complexidade do software é diretamente proporcional à dificuldade de mantê-lo. Porém, nem sempre a complexidade presente em um software é necessária. Isto é, o software poderia ser mais simples do que é. No entanto, há um mito disseminado na comunidade de desenvolvedores, que não é privilégio dela, de que só coisas complicadas têm qualidade.

Segundo Albert Einstein, “qualquer tolo inteligente pode fazer coisas grandes, mais complexas e mais violentas”, mas “é preciso um toque de gênio – e um pouco de coragem – para se mover na direção oposta”. Einstein também afirmou que “a maioria das ideias fundamentais da ciência é essencialmente simples, e pode, como regra, ser expressa em uma linguagem compreensível para todos”. Em consonância com Einstein, Rasmus Lerdorf, o criador do PHP, afirma que “a solução mais complexa raramente é a certa”.

Às vezes, conseguimos resolver um problema muito rápido e ficamos felizes ou até impressionados com isso. É algo muito comum que iniciantes na linguagem PHP consigam criar programas funcionais sem sequer saber exatamente o que estão fazendo. Só que fazer algo rápido pode gerar complexidade, enquanto fazer algo benfeito pode gerar simplicidade. De acordo com o Princípio de Balog, rápido e benfeito são dois modos mutuamente exclusivos de se fazer algo.

Balog foi um colega com quem tive a honra de trabalhar e aprender um pouco sobre controle de processos, e cujo terno peguei emprestado acidentalmente. Sua frase mais marcante era: “Você quer rápido ou benfeito?”.

O fato é que algo criado para resolver um problema pode tornar-se um problema, se for pensado apenas para o momento. Por exemplo, tapar buracos na estrada em vez de recapeá-la não resolve o problema, só o ameniza por um tempo para depois fazê-lo retornar pior.

No entanto, você percebe mesmo que a situação tornou-se crítica quando não consegue mudar algo feito por precisar dele para contornar um problema maior. Sistemas legados criados com tecnologia obsoleta, por exemplo, geralmente são pouco flexíveis e têm alto custo pela limitação do suporte – isso quando não existe apenas um fornecedor do produto ou serviço.

Software e produção

Há um pensamento disseminado entre engenheiros de produção que veem o software como um produto de engenharia similar a um carro, avião ou liquidificador, de que o programador não deve pensar, apenas executar. Nesse pensamento, quem programa é visto como mais uma peça ou mecanismo de uma enorme linha de produção, na qual uma série de operações é repetida de modo interminável para gerar várias instâncias de um projeto de engenharia.

Aqui há um sério problema na interpretação do processo de produção. Na maior parte dos produtos industriais, o problema a ser atacado é a produção do maior número de cópias de um produto no menor espaço de tempo. Só que a reprodução do software é a parte mais fácil que existe, ou seja, o problema central para o software não é a criação das cópias, mas, sim, a criação do original.

Todavia, isso não invalida a aplicação da engenharia na produção de software, pois podemos constatar, pelas empresas que a empregam, que isso gera resultados – embora não possamos dizer que sejam totalmente satisfatórios.

A engenharia trata do processo de construção de software, que envolve a gerência de recursos humanos, hardware, software para gerar software e o temido e inexorável tempo. Mas não podemos nos iludir achando que, se garantirmos a qualidade do processo, o produto naturalmente será de qualidade.

Temos de trabalhar na organização, na estrutura do produto; caso contrário, teremos apenas um processo para produzir rapidamente um software com problemas de manutenção, e o tempo que economizamos no desenvolvimento será uma vaga lembrança quando noites e finais de semana forem perdidos em desesperados esforços para corrigir falhas ou implementar novas funcionalidades.

Copiar, colar e as três moléstias do código

Falaremos mais adiante sobre reaproveitamento de implementações anteriores em novos projetos. Essa é uma forma de aumentar a produtividade e diminuir riscos, mas isso só funciona efetivamente se for feito de uma forma bem organizada. Caso contrário, se tornará uma temível aplicação de POCC – programação orientada a copiar e colar.

Além da POCC, há terríveis moléstias do código-fonte. E três delas são fatais: Dead Code, CBI e TDB.

- Dead Code é o código que não faz absolutamente nada em seu software, mas cresce como uma gangrena que vai matando um membro gradativamente. A pergunta imediata seria: por que se mantém esse código se ele não faz nada? A resposta é simples: se nunca for realizada uma análise de dependências e uma refatoração no código a cada manutenção, um bloco de código que um dia fez alguma coisa se tornará um zumbi dentro do software. E, assim como lixo não recolhido só se acumula, o código que falece após uma manutenção junta-se ao que já estava moribundo. Logo, em pouco tempo, temos mais zumbis que na série de televisão *The Walking Dead*.
- CBI é a sigla para *Cross Bug Injection*. Você corrige um bug em uma versão do software. Em um momento posterior, você adiciona alguma funcionalidade usando POCC e, inexplicavelmente, o bug retorna, como se fosse Jason na série *Sexta-Feira 1 3*.

- TDB é a sigla para *Total Destruction Button*. Essa é a situação na qual os programadores alteram um pequeno trecho de código e o software para de funcionar completamente. É como se aquele trecho provocasse a destruição completa das funcionalidades. Quando os programadores perdem completamente o controle sobre o software ou caem de paraquedas em um projeto sem qualquer documentação, ocorre outra moléstia, que é a síndrome de pânico de TDB.

1.2 Arquitetura de software

Depois da sessão de terror anterior, na qual tentamos incutir medo em seu coração, você deve estar preocupado o suficiente para dar valor à arquitetura de software.

Esse termo que parece ter sido forjado para impressionar amigos durante conversas de bar pode ser explicado pelos seguintes aspectos:

- A arquitetura define os elementos de software.
- Os sistemas podem compreender (e compreendem) mais do que uma estrutura.
- Todo sistema de computação com software tem uma arquitetura de software.
- O comportamento de cada elemento é parte da arquitetura.

Não basta criar uma arquitetura, é necessário criar uma boa arquitetura. Uma boa arquitetura de software reflete a preocupação com a entrega de funcionalidades, que realmente gera valor para o software, e suporta mudança, interação com o usuário final e descoberta e fácil compreensão das funcionalidades. Ou seja, a boa arquitetura é enxuta, entrega o que realmente interessa e é ágil, consegue mudar rapidamente.

Dessa forma, reduz lixo e inconsistência, e exige menos retrabalho. Uma boa arquitetura faz um código-fonte parecer familiar, de modo que podemos chamá-lo de código habitável. Ao trabalhar com o código-fonte de um software, a pessoa que programa deve se sentir tão confortável e à vontade como se estivesse em sua casa – considerando que sua casa seja um lugar agradável, é claro.

Uma decisão importante que deve fazer parte da arquitetura de uma aplicação de software é inspecionar para saber se ela proporcionará o reúso de componentes e como fará isso.

Reúso

Sistemas de software mudam durante seu tempo de vida, e a fase de manutenção é proporcional a esse tempo. Muitos, senão a maioria dos programadores, não trabalham com desenvolvimento de novos softwares, mas com manutenção de software existente. E código existente impõe restrições.

Nesta seção, vamos estabelecer a ligação entre o problema de manutenção de software e a prática da reutilização. Para projetar um sistema que seja robusto em relação a tais mudanças, deve-se levar em conta como o sistema pode necessitar de mudanças ao longo de sua vida. Um projeto que não considera essa possibilidade está sujeito ao risco de uma grande reformulação no futuro.

O reúso de software traz diversos benefícios, como o aumento de confiança no software, a redução de risco do processo, o uso eficiente de especialistas, a conformidade com padrões e o desenvolvimento acelerado.

É claro que, para reusar alguma coisa (que significa usá-la novamente), é preciso pelo menos saber como usá-la. A função é uma das unidades de reúso da programação estruturada. A outra é a sub-rotina, que também é implementada por PHP.

A sub-rotina e a função foram encapsuladas pelas classes na programação orientada a objetos, como o nome de método. O

método é a real menor unidade reutilizável de um programa orientado a objetos, embora esteja subordinado a uma classe.

Padrões

Dessa forma, chega-se à necessidade de padrões para promover o reúso. Padrões de projeto é o termo usado para nos referirmos a uma solução para um problema genérico cuja eficácia foi comprovada pela experiência. Também podemos chamar isso de receita.

A receita tem ingredientes e o modo de fazer, mas não faz exigência de marcas de produto. Da mesma forma, o padrão de projeto traz as orientações para uma implementação sem exigir uma linguagem de programação específica.

Padrões de arquitetura

Há um grupo de padrões de projeto que denominamos de padrões de arquitetura. Esses são os padrões que envolvem lógica de domínio. Esse termo imponente corresponde à letra M do padrão de projeto MVC e refere-se ao trabalho que a aplicação tem de fazer para o domínio de problema com o qual o desenvolvedor está trabalhando.

Padrões de arquitetura são soluções genéricas para problemas que você não resolverá com softwares de prateleira, que estão prontinhos para uso. Você terá de desenvolver uma nova solução e, a princípio, sem vislumbrar quase nenhuma possibilidade de reaproveitamento de implementações anteriores com relação ao negócio.

Criação de camadas

Camadas de software são padrões de arquitetura. Elas constituem soluções genéricas, pois podem ser aplicadas a qualquer software, independente de seu negócio ou finalidade. Em um projeto Laminas,

elas são muito importantes. Você as verá representadas como pastas (ou diretórios) quando iniciarmos nossa implementação no capítulo 6.

Uma camada em um software é uma analogia de uma camada em uma torta ou em um bolo. Talvez o bolo de casamento seja a analogia mais apropriada, pois geralmente é feito com vários bolos, um sobre o outro. Cada camada de software é independente e comunica-se apenas com a camada imediatamente abaixo e acima.

Por exemplo, em um sistema de quatro camadas (figura a seguir), em que 1 é a camada mais alta e 4 a mais baixa, 1 consome serviços de 2, mas ignora a existência de 3 e 4. Poderíamos chamar isso de “regra da ignorância das camadas não adjuntas”, ou RICA.

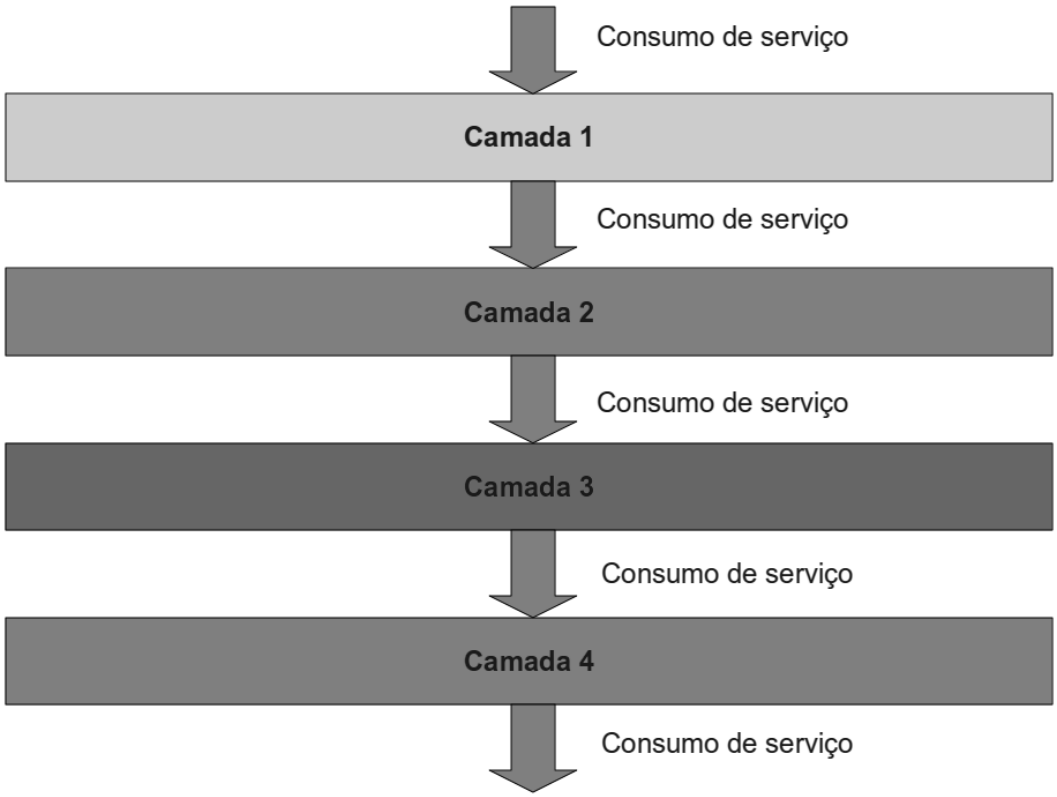


Figura 1.1: Um sistema de quatro camadas

Entretanto, nem todas as arquiteturas de camadas respeitam a RICA. Nesse caso, a metáfora do bolo perde um pouco do significado, e seria mais adequado referir-se a essas outras arquiteturas como arquiteturas de nós (como nós de uma rede de comunicação de dados).

Modularização

Camadas relacionam-se ao conceito cartesiano de divisão de um problema em partes menores. O bolo de casamento é constituído de bolos menores, que podem ser assados separadamente. Um sistema pode ser constituído de módulos, que podem ser desenvolvidos separadamente. O módulo é o ator principal na arquitetura do Laminas. Para o caso de você ter passado direto por aqui da primeira vez, há uma referência a esta seção no capítulo 6, quando estivermos implementando o primeiro projeto.

Em um dos episódios do desenho animado *G.I. Joe* (que no Brasil foi chamado de *Comandos em Ação*), o Comandante Cobra apossou-se de um satélite dotado de um canhão de raios que lhe permitia atingir qualquer local na Terra. Quando os Joes foram atacar a base da Cobra, de onde o Comandante controlava o satélite, com grandes carros de combate blindados, foram alvejados facilmente, pois uma espécie de radar indicava suas coordenadas.

No entanto, os Joes perceberam que os que estavam a pé não eram atingidos, então abandonaram seus veículos e começaram a correr. O radar não podia mostrá-los, pois eram muito pequenos para a precisão do aparelho, então o Comandante Cobra não conseguia mais acertá-los. Conclusão: dividindo-se em unidades de ataque menores, eles conseguiram invadir a base e recuperar o controle do satélite.

A análise consiste na divisão de um problema em partes menores, que sejam mais compreensíveis. A análise de um sistema, portanto, deve produzir naturalmente subsistemas, que podem ser camadas

ou outros tipos de estruturas que tenham interesses ou responsabilidades bem definidas.

A questão da compreensão é um critério de parada no trabalho de divisão. Não dividimos de forma indefinida, parando somente na menor funcionalidade a que se pode chegar. Devemos parar na menor unidade funcional que gere valor, que traduza uma regra de negócio, ou que pelo menos descreva uma tarefa que sirva de apoio para um processo que encontre similar no mundo real, ou que possa ser desenvolvida manualmente com consciência sobre sua finalidade.

Componentes de software podem ser complexos. A complexidade deve-se ao fato de os componentes encerrarem conhecimento suficiente para entregar valor. Um bloco que faz alguma coisa, mas não entrega valor, é como se fosse metade de um bloco de cimento. Você tem de juntá-lo com outro pedaço para ter alguma coisa realmente útil.

As camadas fazem parte de uma estratégia maior para domínio e controle do software. Entretanto, há alguns aspectos negativos desse tipo de modularização. Vamos descobrir por que, mesmo assim, persistiremos em sua utilização.

Arquitetura de referência

Os aspectos negativos do uso de camadas podem ser contornados com a aplicação de outras técnicas. Algo mais grave é a miríade de possibilidades que podem surgir quando se pensa em como dividir uma aplicação em camadas. Isso faz parte do processo de definição da arquitetura de uma aplicação, que envolve (ou pelo menos deveria envolver) todos os patrocinadores (BASS et al., 2003). A diversidade de necessidades e interesses pode resultar em inúmeras combinações, fazendo com que definir a arquitetura de uma aplicação não seja um processo determinístico.

Para uma empresa que desenvolve software, trabalhar com arquiteturas diferentes para cada aplicação significa o aumento da complexidade do controle dos projetos. Arquiteturas diferentes resultam em elementos diferentes de software, ou, mais especificamente, em componentes de software com padrões diferentes. E, como vimos anteriormente, a falta de padrões é um obstáculo para o reúso de software.

Há três conceitos importantes em torno dos padrões de projeto de software: o padrão arquitetural, o modelo de referência e a arquitetura de referência. O padrão arquitetural é uma descrição de um elemento e seus tipos de relação junto a um conjunto de restrições sobre como eles devem ser usados. Um modelo de referência é uma divisão de funcionalidades junto a um fluxo de dados entre as partes.

Uma arquitetura de referência é um modelo de referência mapeado para elementos de software, e os dados que fluem entre eles. Logo, ela é o resultado da combinação do padrão arquitetural com o modelo de referência.

Ela tem uma importância que cresce com o tamanho de uma empresa de desenvolvimento de software. Quanto mais equipes existirem, mais difícil será a comunicação. Isso se complica caso as equipes estejam em locais diferentes (cidades, estados ou até países). É inviável reunir todo mundo sempre que um determinado time tem de decidir sobre a arquitetura de uma aplicação específica.

A arquitetura de referência permite que arquiteturas de software diferentes definam estruturas de aplicações que possuam elementos comuns entre si (componentes reutilizáveis). Ao estabelecer uma arquitetura como padrão, não se perde tempo para criar uma arquitetura do zero, nem para selecionar uma dentre várias. Isso também evita a necessidade de comunicar os envolvidos sempre que um novo sistema for desenvolvido, e a verificação de conformidade tende a ficar mais madura à medida que mais sistemas usarem a mesma arquitetura.

MVC

Em nosso primeiro projeto com Laminas, no capítulo 6, vamos delegar o controle da aplicação para uma classe chamada `Laminas\MVC\Application`. Ela faz parte de um componente que implementa um padrão de arquitetura que se encaixou como uma mão em uma luva em aplicações Web, baseadas na arquitetura cliente-servidor implementada sobre o protocolo HTTP.

Esse padrão de arquitetura é o MVC (*Model-View-Controller*), que define três camadas para uma aplicação de software. Isso não quer dizer que seja possível criar arquiteturas de apenas três camadas para todas as aplicações. Na verdade, significa que podemos iniciar a divisão em camadas buscando essas três responsabilidades.

As três camadas do MVC têm por objetivo criar duas separações principais em uma aplicação: a separação entre a interface com o usuário e os dados, e a separação entre a interface do usuário e o controle da aplicação. Para isso, os papéis definidos assumem as seguintes responsabilidades:

- O **Model** (modelo) é responsável por representar alguma informação sobre o domínio. Ele é um objeto não visual que contém todos os dados e comportamentos, exceto os que são usados pela interface de usuário.
- A **View** (visão) tem como responsabilidade exibir os dados do modelo na interface com o usuário. Várias visões podem ter como fonte de dados um mesmo modelo. Na verdade, a visão é uma forma de apresentação dos dados do modelo.
- O **Controller** (controlador) é o ente que atualiza a visão a partir de uma entrada do usuário, ou quando o modelo sofrer alguma modificação.

A próxima figura ilustra o relacionamento entre os papéis do MVC. Pode-se perceber que, como camadas, os três entes não respeitam a RICA. Isso não impede que elas sejam independentes, porque isso depende da interface de comunicação entre elas.

Se as referências aos objetos que representam cada camada são indiretas, uma camada pode ser substituída por outra, sem que aquela que consome seus serviços perceba a mudança. Esse assunto será retomado quando definirmos a arquitetura da API que este livro propõe.

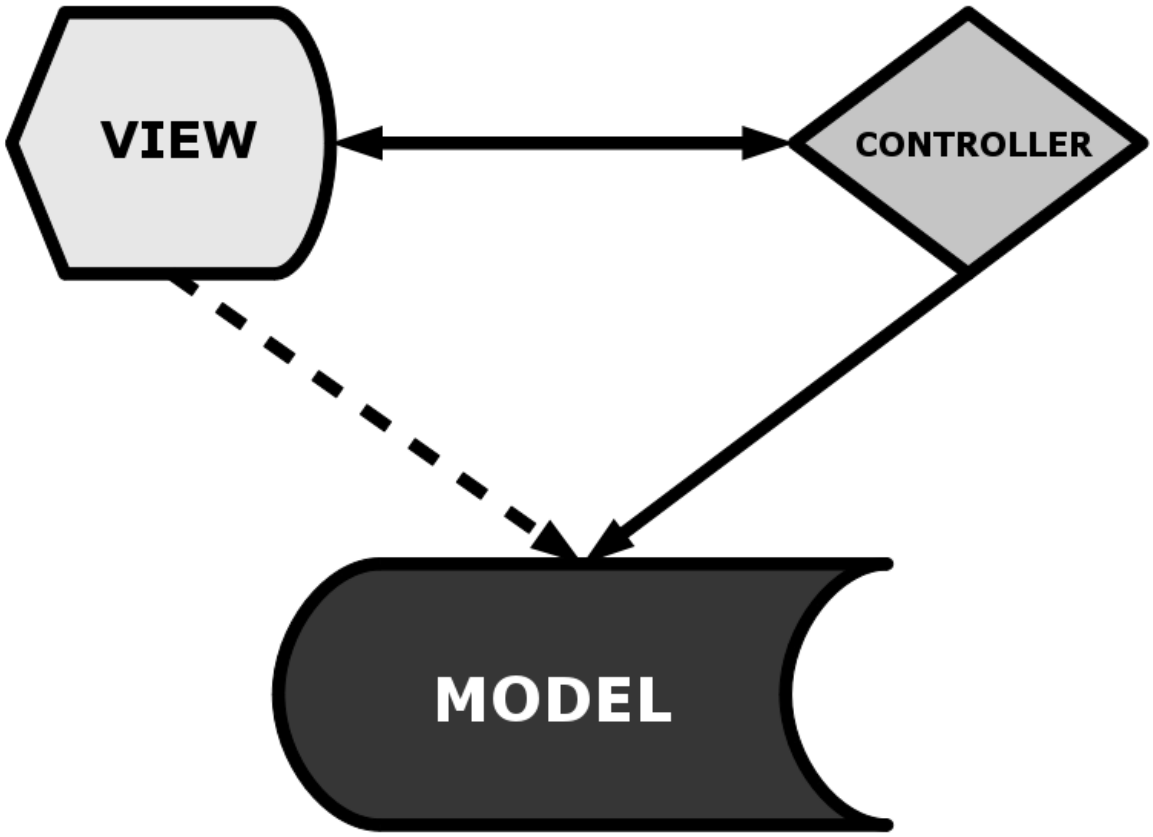


Figura 1.2: Papéis do padrão MVC

Os responsáveis pela programação do modelo não precisam ter conhecimento das apresentações utilizadas. Ao mesmo tempo, os responsáveis pela camada de apresentação não precisam se preocupar com as políticas de negócio e as interações com bancos de dados encapsuladas pelo modelo.

O padrão MVC descreve elementos (model, view e controller) e seus tipos de relação (a visão depende do modelo, o controlador manipula o modelo e atualiza a visão), além de estabelecer um

conjunto de restrições sobre como eles devem ser usados: a visão trata da interface com o usuário, o modelo cuida das regras de negócio e o controlador manipula ambos. Logo, pode-se afirmar que o MVC é um padrão arquitetural.

O MVC também faz uma divisão de funcionalidades - interface com o usuário, regras de negócio, controle de entrada e saída - e define um fluxo de dados entre as partes: a entrada da visão passa pelo controlador para ir ao modelo, mas os dados do modelo podem ir diretamente para a visão. Logo, MVC também é um modelo de referência.

Como um padrão arquitetural combinado com um modelo de referência constitui uma arquitetura de referência, pode-se dizer que o padrão MVC estabelece uma arquitetura de referência. Deve-se ressaltar que, conforme foi descrito anteriormente, a arquitetura de referência não impede que a arquitetura específica de uma aplicação contenha outros elementos arquiteturais. Ela é apenas o ponto de partida, o norte da bússola.

Frameworks

Supondo que um problema genérico é um conjunto de problemas específicos, a solução de um problema genérico deve ser uma combinação de padrões de projeto. O MVC, enquanto padrão de projeto, é apenas parte da solução. Deve ser combinado com outros padrões, conforme as necessidades do projeto. A combinação da implementação de vários padrões de projeto constitui o que denominamos de framework.

A inversão de controle estabelece a diferença entre toolkits ou bibliotecas de sub-rotinas e frameworks. Nas primeiras, o desenvolvedor escreve o corpo principal da aplicação e chama o código que quer reutilizar. No último, ele reutiliza o corpo principal e escreve o código que o framework chama. Você verá a inversão de controle no arquivo `index.php` do primeiro projeto Laminas que implementará no capítulo 6.

A fase de manutenção é a mais longa no ciclo de vida do software, por isso precisamos dar-lhe atenção especial. Quanto maior um sistema, mais complexo ele fica. E quanto mais complexo, mais difícil de controlar. O que significa que não temos uma ideia precisa do impacto que uma mudança pode causar no sistema inteiro.

A reusabilidade máxima antecipa as mudanças. Como foi visto, para sistemas orientados a objeto, isso é obtido com a adoção de frameworks. É o melhor meio de obter a máxima reusabilidade, antecipar as mudanças e facilitar a manutenção do software. Observe que não falamos de construção de framework, mas de adoção. Um conselho de baixo custo é: utilize um framework que seja software livre e aberto - como o Laminas.

1.3 Conclusão

Você aprenderá a usar um framework porque ele é a implementação de vários padrões de projeto, e cada padrão de projeto é a melhor solução encontrada para um problema recorrente. Usar um framework significa reaproveitar conhecimento e experiência de outros programadores e deixar de cometer erros que outras pessoas já cometeram.

Capítulo 2
Boas práticas de desenvolvimento

Os que se encantam com a prática sem a ciência são como os timoneiros que entram no navio sem timão nem bússola, nunca tendo certeza do seu destino. – Leonardo da Vinci

Pelo fato de PHP ser muito fácil de aprender, ou pelo menos de ser muito fácil gerar resultados rapidamente pela aplicação da POCC, pessoas sem formação adequada acabam produzindo verdadeiros monstros de Frankenstein com PHP. Um pedaço de código copiado dali mais outro pedaço copiado de lá, um tema roubado de uma aplicação bonitinha e pronto: rapidamente temos um sistema de informação que, nos primeiros dias, faz a plateia dizer: *Ohhhh!*

Só que, alguns meses depois, essa plateia já o está vaiando, depois de perceber que as mudanças solicitadas não são implementadas no tempo esperado; ou que, quando são, fazem com que outras funcionalidades sejam neutralizadas.

Existe uma diferença bem grande entre o desenvolvedor PHP profissional e o paraquedista PHP, que é a pessoa que se viu repentinamente trabalhando com uma aplicação PHP, mas não sabe exatamente o que está fazendo. Imagine um paraquedista que acaba caindo dentro de um acampamento inimigo e se vê cercado. Ele pode se render ou sair atirando para todos os lados.

Isso pode realmente ocorrer na manutenção de uma aplicação PHP. Mas o pior é quando acontece na construção, o que significa que os envolvidos no desenvolvimento estão realmente perdidos.

Sem uma formação e orientação adequadas, um técnico que se mete a desenvolver em PHP, mesmo que já tenha experiência com outra linguagem, acaba cometendo verdadeiros crimes de programação.

Rasmus Lerdorf, o criador do PHP, declarou que essa linguagem não se trata de uma implementação sobre pureza de princípios de ciência da computação ou arquitetura, mas sobre resolver problemas da web de uma forma que é admitidamente feia, porém extremamente funcional e conveniente.

Isso parece meio contraditório em relação a todo o discurso que fizemos até agora. Ribeiro, autor de *PHP Jedi* (2009), afirma que beleza nem sempre é fundamental, mas um código bem escrito faz toda a diferença. De acordo com ele, os programadores PHP devem adotar boas práticas para criar um código de qualidade.

Se você já é um PHP Jedi, pode ir direto para o ambiente de desenvolvimento. Porém, se ainda é um PHP Padawan, é altamente recomendável que leia as recomendações desta seção. A não ser que você queira seguir o plano de carreira PHP Sith, que consiste em três estágios:

- **Aprendiz –** O programador pesquisa... no Google. Ele aplica metodologia... *CCF? B!* (Copiou. Colou. Funcionou? Beleza!)
- **Cavaleiro –** O programador ainda pensa... em como se livrar do problema o mais rápido possível. Ele aplica técnicas de POG (Programação Orientada a Gambiarra). A gambiarra é para o padrão de projeto o que a antimatéria é para a matéria. Se existe um modo adequado de resolver uma complicação, a POG certamente terá um modo inadequado equivalente, que criará outro sufoco, porque problema atrai problema na proporção direta do produto de suas complexidades.
- **Mestre –** O programador já não pensa, ele atira. POG para ele é coisa de fraco, ele usa a metodologia XGH (*eXtreme Go Horse*). Nesse estágio, o desenvolvedor reconhece apenas que existem três modos de resolver um problema: o certo, o errado e o modo XGH, que é igual ao errado, só que mais rápido. Quer dizer, antes ele até tinha noção do que estava fazendo (que já era errado), agora ele faz algo errado e nem tem ideia de como funciona.

O trabalho de quem programa não é algo trivial e não deve ser menosprezado. É preciso trilhar o caminho do Jedi com paciência para aprender cada etapa, em vez de copiar desesperadamente soluções paliativas.

Este é um livro sobre um framework que vai facilitar o desenvolvimento de aplicações PHP, e não gerar código automaticamente. Algumas ferramentas vendem soluções de geração automática de código, mas geram apenas coisas extremamente repetitivas.

Um exemplo é caso da plataforma de desenvolvimento do programa gerador do imposto de renda: é sempre a mesma coisa. Esse tipo de solução está baseado em uma zona de conforto na qual há poucas solicitações e que não serve para tratar de requisitos mutáveis.

Focar no resultado não significa fazer qualquer coisa que funcione. Você tem de criar algo funcional, mas bem estruturado, por mais simples que seja essa estrutura. Não diga que vai arrumar mais tarde, porque, na prática, mais tarde significa nunca.

Vamos agora para as recomendações que evitarão que você seja consumido pelo lado negro da força do PHP.

2.1 Princípios da boa programação

Aqui reúno princípios citados por vários autores e que se encontram dispersos em vários artigos e livros. Todos devem ser seguidos se você quiser desenvolver aplicações com Laminas da forma adequada. São estes:

- **Princípio da abstração:** cada pedaço significativo de funcionalidade em um programa deve ser implementado em apenas um lugar no código-fonte.

- **DRY - Don’t Repeat Yourself (Não repita a si mesmo):** se você identificar um bloco de código que se repete em vários lugares, modularize-o, criando uma função, um método, uma classe ou um trait. A aplicação deste princípio garante o anterior.
- **KISS - Keep it simple, stupid! (Mantenha isso simples, estúpido!):** a simplicidade deve ser sempre o objetivo principal. Código simples leva menos tempo para ser escrito, tem menos bugs e é mais fácil de modificar. Lembre-se de Rasmus Lerdorf: o importante não é a beleza, é a funcionalidade.
- **Evite criar um YAGNI - You aren’t going to need it! (Você não precisará disso!):** não adicione funcionalidade até precisar dela.
- **Não me faça pensar:** se um código exige horas de estudo e reflexão para ser compreendido, então ele provavelmente precisa ser simplificado.
- **Princípio aberto/fechado:** entidades de software (classes, módulos, funções etc.) devem ser abertas para extensão, mas fechadas para modificação. Em outras palavras, não escreva classes que as pessoas possam modificar, mas que possam estender.
- **Princípio da perplexidade mínima:** o código não deve surpreender a pessoa que vai lê-lo (que provavelmente será o mantenedor). Isso implica em adotar padrões e convenções que sejam bem conhecidos e facilitem a compreensão da estrutura.
- **Princípio da responsabilidade única:** um componente de código, que pode ser uma classe ou função, por exemplo, deve executar uma única e bem definida tarefa.
- **Maximize a coesão:** funcionalidades similares devem estar dentro do mesmo componente de software.
- **Minimize o acoplamento:** os componentes de código de um software devem ter mínima dependência de outras partes. Isso evita o efeito *Botão da Destruição Total* e melhora o reúso.
- **Oculte os detalhes da implementação:** isso não quer dizer que você deva criptografar o código. A complexidade do código pode ser diminuída com a delegação de tarefas para outros

componentes, aplicando a modularização em um módulo. O componente fica mais compreensível, como também fica mais fácil alterar uma parte da implementação sem modificar o componente de nível mais alto, que se relaciona com a aplicação.

- **Lei de Deméter:** na verdade, é um princípio geral aplicado a software, baseado na frase “fale somente com seus amigos”, que poderia também ser expressada pela recomendação paterna “não fale com estranhos”. Sua aplicação a software implica que componentes de código devem apenas se comunicar com suas relações diretas (por exemplo, classes com suas superclasses, com os objetos que elas contêm e com objetos passados como argumentos). Isso garante o baixo acoplamento.
- **Evite a otimização prematura:** uma insanidade frequente é a ideia de criar algo que seja ótimo na primeira iteração ou versão do software. Primeiro, você cria um código que funciona; depois, trabalha para torná-lo mais rápido. De acordo com Donald Knuth, citado por Diggins (2011), 97% do tempo de desenvolvimento é gasto na implementação de pequenas melhorias de performance. O fato é que não se pode aperfeiçoar o que ainda não existe. Primeiro, implemente, coloque em produção, meça e então faça a otimização.
- **Reúso de código é bom:** falamos sobre isso anteriormente.
- **Separação de interesses:** esta é a expressão culta de “cada um com seus problemas”. Cada módulo de código é responsável por diferentes áreas de funcionalidade, e nenhuma área deve interferir nos interesses da outra. Se um módulo precisa da funcionalidade de outro para completar ou prosseguir uma tarefa, deve se comunicar com ele. Em programação orientada a objetos, ninguém faz tudo; cada um faz somente a sua parte.
- **Abrace a mudança:** se você quiser trabalhar com software, saiba que não existe zona de conforto, porque a mudança é a única certeza que terá. A tecnologia muda e os requisitos mudam, logo, o software terá sempre de mudar. Software não é

> um monumento que deve ser projetado para resistir ao tempo e ser contemplado pelas gerações futuras como um memorável empreendimento dos ancestrais, mas, sim, como um organismo que deve se adaptar caso queira sobreviver às intempéries da evolução política, econômica, social e tecnológica.

Se ainda restou alguma dúvida sobre toda essa teoria, não se preocupe, pois aplicaremos esses princípios ao longo deste livro.

2.2 Calistenia de objetos

Ao usar Laminas, você está abraçando a programação orientada a objetos. Mas temos de reconhecer que o bom projeto orientado a objetos é difícil de aprender. A POO proporciona um nível de reúso maior do que a programação procedural, certamente, mas somente se bem aplicada.

Os paradigmas de programação são resultado da evolução combinada do pensamento dos desenvolvedores de software e dos cientistas da computação. Eles não extinguem um ao outro, mas são fundados em seus precedentes. Assim, quando você programa com Orientação a Objetos, também está programando de forma procedural; mesmo que esse paradigma esteja aplicado em uma parte menor do software.

O bom projeto orientado a objeto é coeso, tem baixo acoplamento, nenhuma redundância, encapsulamento, testabilidade, legibilidade e foco. A questão é como programar de forma a garantir essas qualidades.

Na verdade, há uma proposta para isso, chamada Calistenia de Objetos. São nove regras que devem ser seguidas para que o código-fonte fique em uma boa forma, como se fosse um atleta com um corpo em que músculos e massa estão em equilíbrio. Neste

caso, a boa forma refere-se a uma forma orientada a objetos que equilibra organização com concisão.

As regras da Calistenia de Objetos são estas:

1. **Um nível de indentação por método –** Essa regra serve para ajudar a diminuir o tamanho de métodos. Dentro de um método, você pode ter uma estrutura de controle ou repetição que tem outra estrutura de controle ou repetição, e isso pode continuar de modo que seu método fique muito grande e ilegível. Então, se você tem um segundo nível de endentação dentro de um método, isso é um sinal de que o código nesse nível pode ser extraído para outro método.

2. **Não use a palavra-chave ELSE –** Não é porque existe a estrutura SE... ENTÃO... SENÃO que você tem de implementar o *senão*. Existem casos em que, se a condição for atendida, o restante do método não será executado. Não há necessidade de *senão*, você simplesmente escapa do método – no caso do PHP, usando a palavra-chave *return*.

3. **Encapsule todas as primitivas e strings –** No caso de PHP, string é um tipo primitivo, como *integer*, *float* e *boolean*. Os tipos primitivos não são objetos, portanto, não possuem atributos nem métodos. Há casos em que você passa vários tipos primitivos como argumentos para um método, e todos esses argumentos estão relacionados. Além disso, é necessário verificar o conteúdo desses argumentos e sua validade. Isso pode ser facilitado se estes forem reunidos como atributos em um objeto que disponha de métodos para responder a todos os questionamentos necessários. Ademais, o tipo classe obriga o PHP a assegurar o conteúdo do argumento, enquanto o tipo primitivo não tem declaração explícita de tipo.

4. **Coleções de classes em primeiro lugar –** Essa regra implica que uma classe que contenha uma coleção de objetos não deve ter outras variáveis-membros. Isso significa que cada

coleção ficará encapsulada em sua própria classe, de modo que comportamentos comuns a uma coleção fiquem em um único lugar, o que segue o Princípio da Abstração.

5. **Um ponto por linha –** No caso do PHP, podemos ter duas interpretações para essa regra:

    i. A primeira, falsa, mas válida, é de que, na verdade, seria um ponto e vírgula por linha. O ponto e vírgula é o separador de instruções do PHP, de modo que você pode escrever inúmeras instruções em uma única linha sem prejuízo da funcionalidade do código. Isso pode ser interessante em competições, mas é uma desgraça para a legibilidade.

    ii. A outra interpretação, que é o significado real dessa regra, é que na verdade seria uma seta por linha. O operador de chamada de métodos no PHP é a seta representada por um hífen seguido de um sinal de maior (`->`). Um método pode retornar como resultado uma referência a um objeto, a partir do qual podemos chamar um método que também retorne uma referência a um objeto. Assim, temos uma sequência de setas em uma única linha em vez de fazer uma chamada a cada linha. Isso pode ser conveniente, às vezes, mas, em outras, pode atrapalhar a legibilidade. Linhas muito extensas ficam difíceis de entender.

6. **Não abrevie –** Variáveis com nomes `$a` , `$b` e `$c` não dizem nada. Métodos com nomes como `do()` também não são claros. Isso não significa que você precisa criar nomes extensos para atributos, métodos e classes, pois isso pode gerar outro problema: duplicação de nomes. Por exemplo, uma classe chamada `Money` não precisa ter um método chamado `convertMoney()` , apenas `convert()` . O que ela não pode ter é um método `cvt()` .

7. **Mantenha todas as entidades pequenas –** Para o PHP, isso implica tentar manter as classes com cinquenta linhas no máximo, e cada namespace com um máximo de dez arquivos (que podem conter classes, interfaces ou traits).

8. **Nenhuma classe deve ter mais de duas variáveis de instância –** Essa regra não quer dizer que você deva fazer um milagre e criar classes com apenas dois atributos. Isso significa que você deve procurar evitar que a maioria dos atributos da classe seja de um tipo primitivo. Atributos não são declarados em uma classe à toa. Você deve utilizá-los para alguma coisa. E, quando uma classe tem muitos atributos, ela provavelmente encerra uma lógica de negócio complexa. Se a classe possui muitos métodos para tratar seus atributos sem que isso seja seu interesse principal, significa que a maioria dos atributos tem um grande potencial para se tornar uma classe separada. Essa regra nada mais é do que uma dica de quando aplicar decomposição de classes.

9. **Nada de getters e setters –** O encapsulamento é uma das características da Orientação a Objetos. Ele dá uma receita básica para criar classes: atributos privados e métodos públicos. Os dados ficam protegidos dentro do objeto e só são modificados através de mensagens, que são as invocações de métodos por outros objetos. Isso realmente é aplicável se os métodos públicos agregarem valor aos atributos privados (ou protegidos), porque, se eles não fizerem absolutamente nada além de ler e gravar, serão inúteis. Para que criar um método para ler um atributo se não será feita nenhuma modificação em seu conteúdo? Da mesma forma, para que criar um método para modificar um atributo se o valor passado como argumento será atribuído ao atributo (com propriedade) sem qualquer alteração? Deixe o atributo público, a menos que você queira tornar sua classe mais complexa. Não precisa pedir “por favor, pode me dar o valor”. Diga “o valor, caramba!”.

Veremos esses princípios em prática ao longo do livro, quando implementarmos nosso projeto.

2.3 Recomendações para desenvolver em PHP

Código habitável só pode ser produzido se convenções de escrita forem seguidas. Comunicação não é algo que flui naturalmente apenas porque as duas partes falam o mesmo idioma. Se fosse assim, qualquer um entenderia letra de médico. Também não basta usar tão somente os vocábulos disponíveis, é preciso dispô-los de forma a facilitar a compreensão. Por isso, usamos parágrafos e saltos de linha quando escrevemos textos.

A primeira grande iniciativa de convenção de escrita de código PHP surgiu com o projeto PEAR (*PHP Extension and Application Repository*), que definiu padrões para a criação de componentes distribuídos pelo seu sistema. Outra iniciativa foi do projeto Zend Framework, que antecedeu o Laminas, o qual publicou em seu guia de referência um detalhado guia de padrões de codificação.

Padrões são legais, mas só funcionam de verdade quando são usados por todos os interessados (ou pela maioria). Vários frameworks PHP foram desenvolvidos, mas cada um inicialmente utilizava seu próprio padrão, o que dificultava a integração. Para serem usados por um time, padrões têm de ser definidos em grupo.

Esse é o caso das recomendações propostas pelo PHP-FIG (*PHP Framework Interoperability Group*). O PHP-FIG escreve PSRs (*Proposing a Specification Request*), que não são resoluções de padrões da linguagem PHP, mas recomendações para desenvolvimento PHP orientado a objetos. Você pode ter acesso a todas as PSRs neste endereço: http://www.php-fig.org/psr. Não se preocupe em ler as recomendações neste momento, pois, ao usar Laminas, você acabará utilizando algumas delas.

Agora que falamos sobre boas práticas de desenvolvimento para aplicações orientadas a objeto, passaremos para a montagem do ambiente de desenvolvimento dos projetos que serão construídos ao longo deste livro.

Capítulo 3
Bússola do ambiente de desenvolvimento

O que nós somos é o que fazemos, e o que fazemos é o que o ambiente nos faz fazer. – John Watson

As ferramentas que constituem o ambiente de desenvolvimento – no qual os exemplos deste livro foram criados – foram adotadas para atender aos seguintes requisitos: abertura, independência de plataforma e gratuidade. Precisamos definir uma configuração padrão que sirva de referência para o seu ambiente de desenvolvimento.

Fique à vontade para executar os exemplos em um ambiente similar, com Apache, PHP e MySQL. Mas, caso ocorra algum problema, sugiro que instale o pacote de software descrito a seguir, e efetue uma comparação de arquivos e seus conteúdos.

3.1 Apache, MySQL e PHP

Apache Friends é uma organização sem fins lucrativos, criada para promover o uso do servidor web Apache, por meio de atividades que tornam mais amigáveis a instalação do software e a leitura da documentação, além da criação de uma comunidade online para ajudar os usuários de Apache. O projeto foi criado em 2002, por Kai 'Oswald' Seidler e Kay Vogelgesang.

O principal produto gerado por essa organização é o XAMPP, que possui distribuições para GNU/Linux, Mac OS X e Windows. O XAMPP é um pacote de softwares que inclui principalmente Apache, MySQL, PHP e PEAR (o X refere-se ao sistema operacional, e as demais letras são iniciais desses softwares). O site do projeto é: https://www.apachefriends.org/pt_br/index.html.

Nesta página, clique no botão de `Download` para selecionar o instalador adequado para seu sistema operacional. Neste livro, foi utilizada uma instalação de XAMPP com PHP 7.

Baixe o instalador e execute-o. Ele abrirá uma interface gráfica que o conduzirá na instalação. XAMPP dispõe de um painel de controle gráfico que permite iniciar e parar os serviços do Apache e MySQL, os dois softwares essenciais do pacote para nós. A seguir, apresentamos a sequência de telas da instalação do XAMPP.

Figura 3.1: Tela inicial do instalador do XAMPP

Setup
Select Components
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
XAMPP Core Files
XAMPP Developer Files
Click on a component to get a detailed description
XAMPP Installer
< Back
Next >
Cancel

Figura 3.2: Seleção de componentes do XAMPP a serem instalados

A partir da tela da figura anterior, basta prosseguir clicando em `Next` até surgir o painel de controle da figura a seguir.

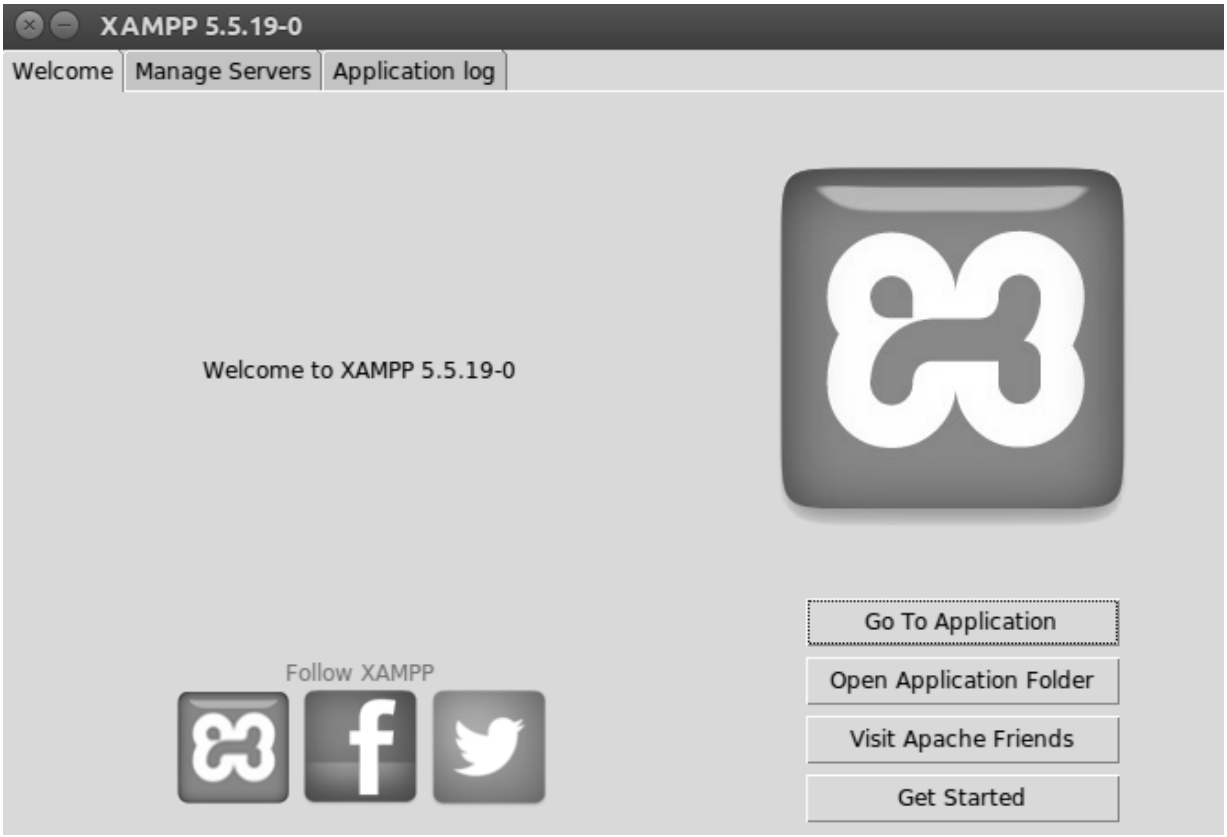


Figura 3.3: Aba Welcome do painel de controle do XAMPP

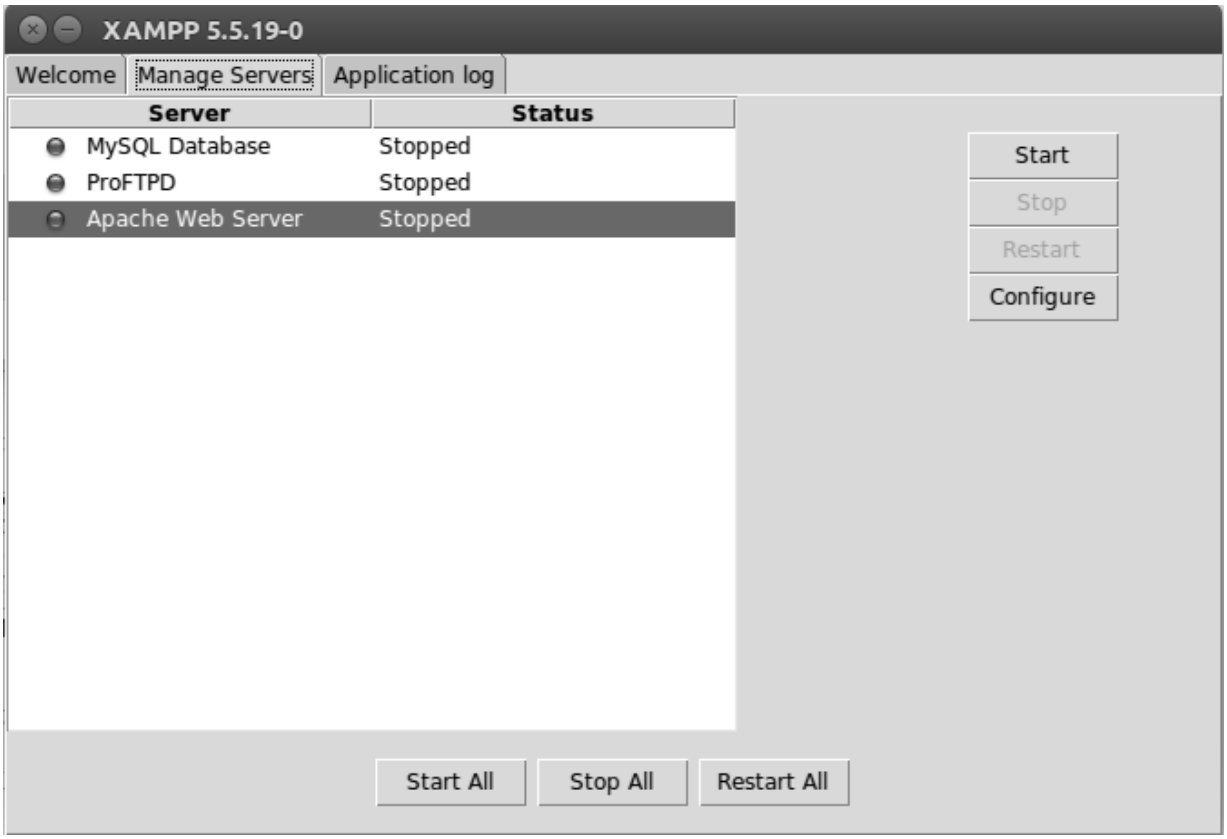


Figura 3.4: Aba de gerenciamento dos serviços do XAMPP

3.2 Ambiente integrado de desenvolvimento

O PDT (*PHP Development Tools*) é um dos inúmeros subprojetos baseados no Eclipse, uma plataforma livre e aberta de desenvolvimento mantida pela Fundação Eclipse. O PDT tem como objetivo criar ferramentas que ajudem os programadores PHP a se tornarem mais produtivos com o uso do ambiente integrado de desenvolvimento Eclipse.

Ele consiste basicamente de um conjunto de plugins que dotam o Eclipse das seguintes funcionalidades: edição de código-fonte PHP, JavaScript e HTML; sintaxe colorida e destaque; completamento de código-fonte; modelos prontos de estruturas de controle; e

autoformatação. Além disso, você pode fazer depuração local e remota usando XDebug ou Zend Debugger.

A instalação do PDT é muito simples. Aqui utilizamos a versão *4.1 5.0 (2020-03)*, disponível em https://www.eclipse.org/pdt/#download. Basta descompactar o arquivo em seu diretório de usuário e criar um atalho em sua área de trabalho. Eclipse é baseado em Java, por isso você precisa da máquina virtual Java instalada em sua máquina ou, mais precisamente, o kit de desenvolvimento Java (JDK). Se não tiver o JDK, baixe a versão mais recente em http://www.java.com/pt_BR/download. E não se esqueça de instalá-lo também.

Crie um atalho para o executável do Eclipse. Ele está no diretório `eclipse-php` e chama-se `eclipse` no GNU/Linux, e `eclipse.exe` no Windows.

Se quiser executar o Eclipse na linha de comando, fique à vontade. O que importa agora é iniciar o Eclipse para realizar as demais configurações de nosso ambiente. Ao iniciá-lo, após a Splash Screen, a aplicação solicitará o workspace, conforme vemos na próxima imagem. O workspace é o caminho de diretório no qual os projetos serão hospedados.

Você pode usar quantos workspaces quiser; o que interessa para o Eclipse é saber qual é o workspace que você quer usar neste momento. O nosso será o diretório de páginas web, neste caso, `/opt/lampp/htdocs` (GNU/Linux) ou `c:\xampp\htdocs` (Windows), veja na figura a seguir. Marque a caixa de verificação `Use this as default and do not ask again` para não vermos mais essa janela.

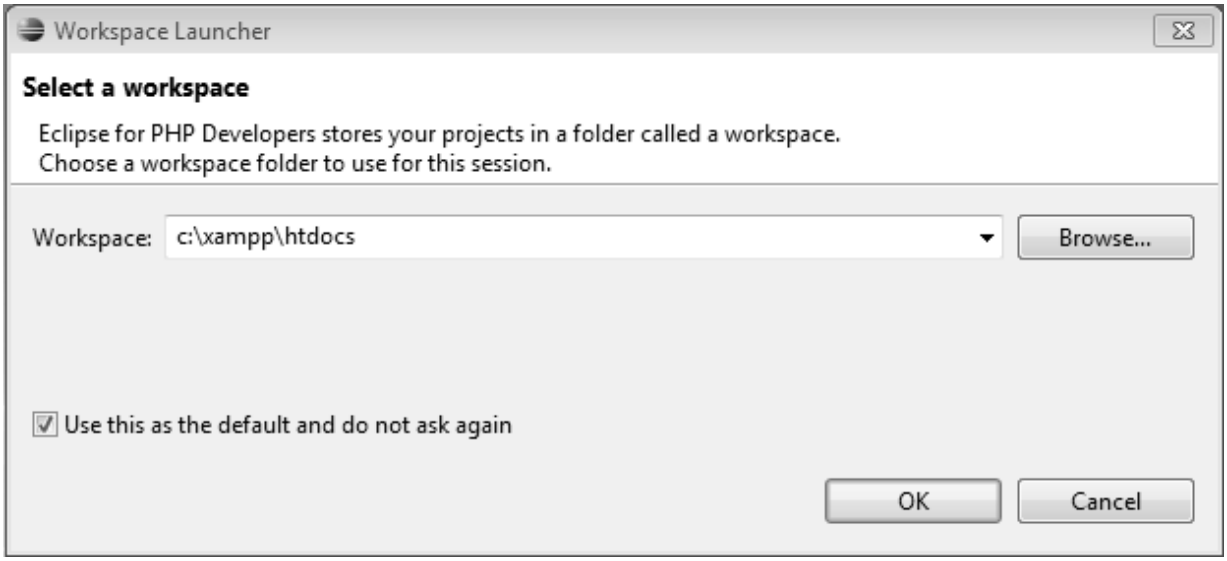

Workspace Launcher
Select a workspace
Eclipse for PHP Developers stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.
Workspace:
c:\xampp\htdocs
Browse...
Use this as the default and do not ask again
OK
Cancel

Figura 3.5: Configuração de workspace do Eclipse

Se você for afobado, deve ter visto uma mensagem de erro `Workspace cannot be created` (figura a seguir). Isso ocorreu porque o seu usuário não tem permissão de escrita nesse diretório. Altere a permissão e tente novamente.

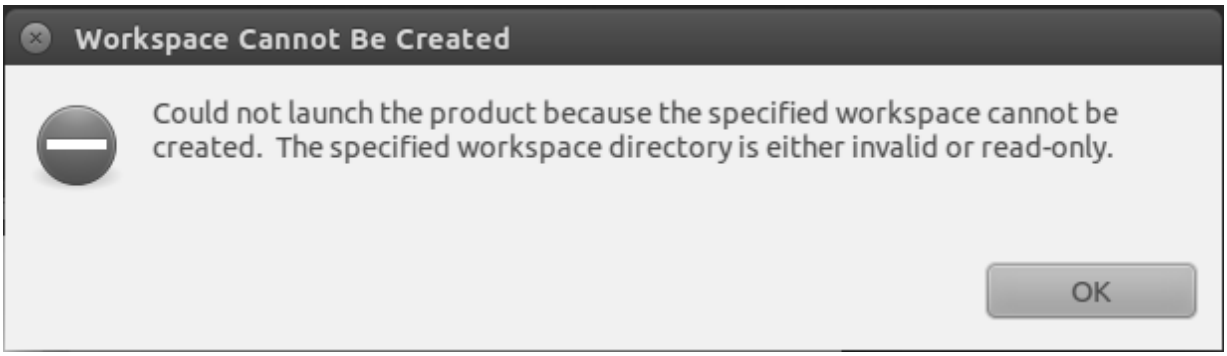

Workspace Cannot Be Created
Could not launch the product because the specified workspace cannot be created. The specified workspace directory is either invalid or read-only.
OK

Figura 3.6: Workspace sem permissão de escrita

O Eclipse será aberto com a perspectiva PHP e a visão `Welcome` ocupando a área central estendida para a direita (figura a seguir). No Eclipse, perspectiva é um conjunto de visões e uma visão, por sua vez, é uma janela interna. Apesar de cada visão fazer parte de um determinado plugin, você pode abrir visões de plugins diferentes e construir a sua própria perspectiva. É possível fechar qualquer visão clicando no X de sua barra de títulos.

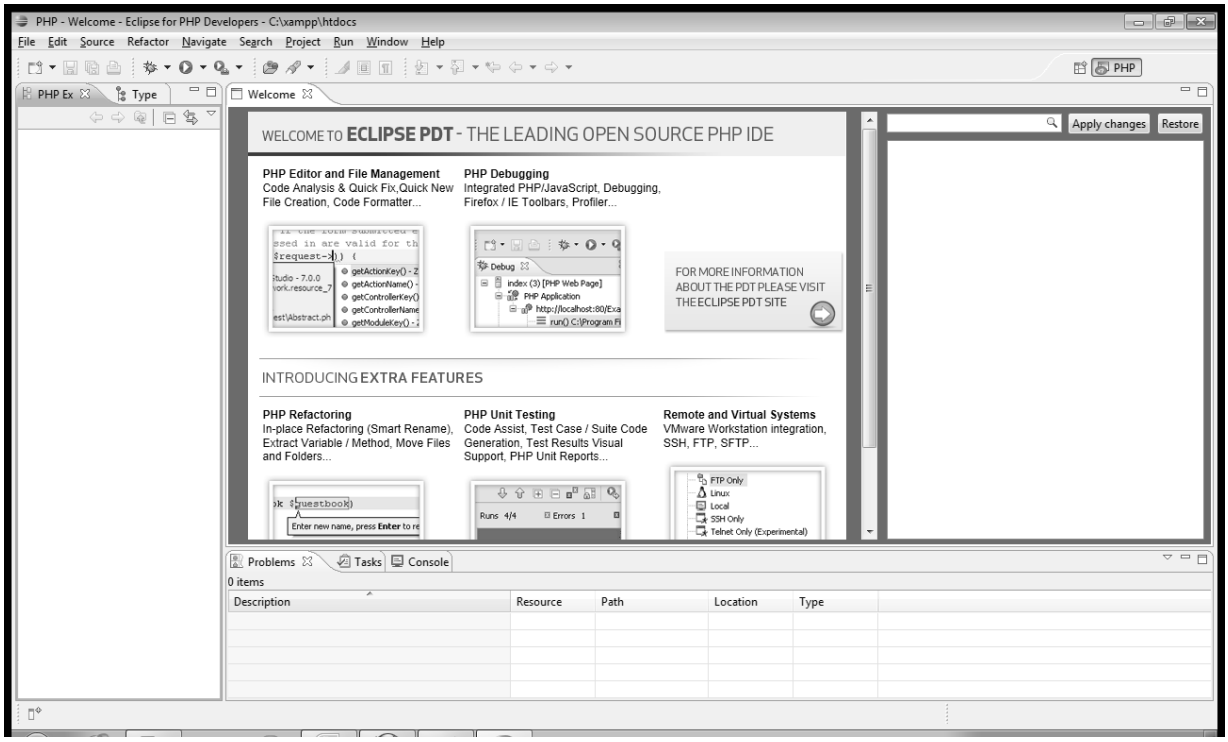

PHP - Welcome - Eclipse for PHP Developers - C:\xampp\htdocs
File Edit Source Refactor Navigate Search Project Run Window Help
PHP
PHP Ex
Type
Welcome
WELCOME TO ECLIPSE PDT - THE LEADING OPEN SOURCE PHP IDE
PHP Editor and File Management
Code Analysis & Quick Fix,Quick New File Creation, Code Formatter...
PHP Debugging
Integrated PHP/JavaScript, Debugging, Firefox / IE Toolbars, Profiler...
FOR MORE INFORMATION ABOUT THE PDT PLEASE VISIT THE ECLIPSE PDT SITE
Apply changes
Restore
INTRODUCING EXTRA FEATURES
PHP Refactoring
In-place Refactoring (Smart Rename), Extract Variable / Method, Move Files and Folders...
PHP Unit Testing
Code Assist, Test Case / Suite Code Generation, Test Results Visual Support, PHP Unit Reports...
Remote and Virtual Systems
VMware Workstation integration, SSH, FTP, SFTP...
FTP Only
Linux
Local
SSH Only
Telnet Only (Experimental)
Problems
Tasks
Console
0 items
Description
Resource
Path
Location
Type

Figura 3.7: Perspectiva PHP com visão Welcome

As visões que você mais utilizará no Eclipse PDT são as seguintes:

Visão	Descrição
PHP Explorer	Exibe os projetos PHP e seus arquivos em uma estrutura de árvore.
Outline	Exibe a estrutura do conteúdo do arquivo que tem o foco no editor de código-fonte. Ao selecionar um item em Outline, o foco é imediatamente movido para a linha correspondente no editor de código-fonte.
Problems	Exibe mensagens de erro (vermelhas) e alerta (amarelas).
Console	Exibe a saída de programas executados pelo Eclipse.

No editor de código-fonte, as linhas são exibidas por padrão. Mas você pode habilitar ou desabilitar essa informação clicando na primeira coluna à esquerda do editor (parte branca). Se após algum tempo o Eclipse apresentar mensagens de erro referentes ao workspace, pode ser que a configuração tenha sido corrompida.

Essa corrupção pode ser decorrente de alterações incompletas na pasta `.metadata`, ocorridas durante instalações ou atualizações de plugins. Nesse caso, inicie o Eclipse com a opção `--clean`. Isso ajuda a corrigir falhas de plugins e melhorar a estabilidade do ambiente.

O tamanho de memória usado pelo Eclipse PDT é configurado no arquivo `eclipse-php.ini`. Se for apresentada alguma mensagem de erro informando que o limite de memória foi alcançado, você deve aumentar o limite máximo alterando o valor da diretiva `-Xmx`. Se a mensagem de erro for sobre a memória `PermGen` (*permanent generation*), aumente o valor da diretiva `XXMaxPermSize`.

Os valores devem ser aumentados em termos de potências de 2. Assim, se o valor atual for 256, por exemplo, aumente para 512.

Algumas teclas de atalho úteis no Eclipse PDT:

Teclas de atalho	Descrição
`F1`	Help ? Dynamic Help.
`F2`	Renomeia o item em Project Explorer e exibe a documentação do elemento selecionado no editor de código-fonte.
`F3`	Navigate ? Open Declaration.
`F4`	Navigate ? Open Type Hierarchy.
`F10`	File ? New.

Teclas de atalho	Descrição
F11	Run ? Debug.
F12	Coloca o foco no editor de código-fonte.
CTRL+1	Correção rápida.
CTRL+ESPAÇO	Assistente de código-fonte.
CTRL+SHIFT+R	Assistente de código-fonte.

Segundo a PSR-2, a indentação deve ser feita com quatro espaços. Logo, se você clicar quatro vezes no espaço para endentar cada nível, seus polegares ficarão mais musculosos. Você pode evitar esse trabalho fazendo com que o Eclipse inclua os espaços quando você pressionar a tecla `TAB`.

No Eclipse, entre no menu `Window` e depois em `Preferences`. Na estrutura em árvore, à esquerda, abra o item `General`; em seguida, `Editors`; e, finalmente, `Text Editors` (figura a seguir). Marque a caixa de seleção `Insert space for tabs`. Por padrão, o item `Displayed tab width` já traz o valor 4, mas é bom confirmar.

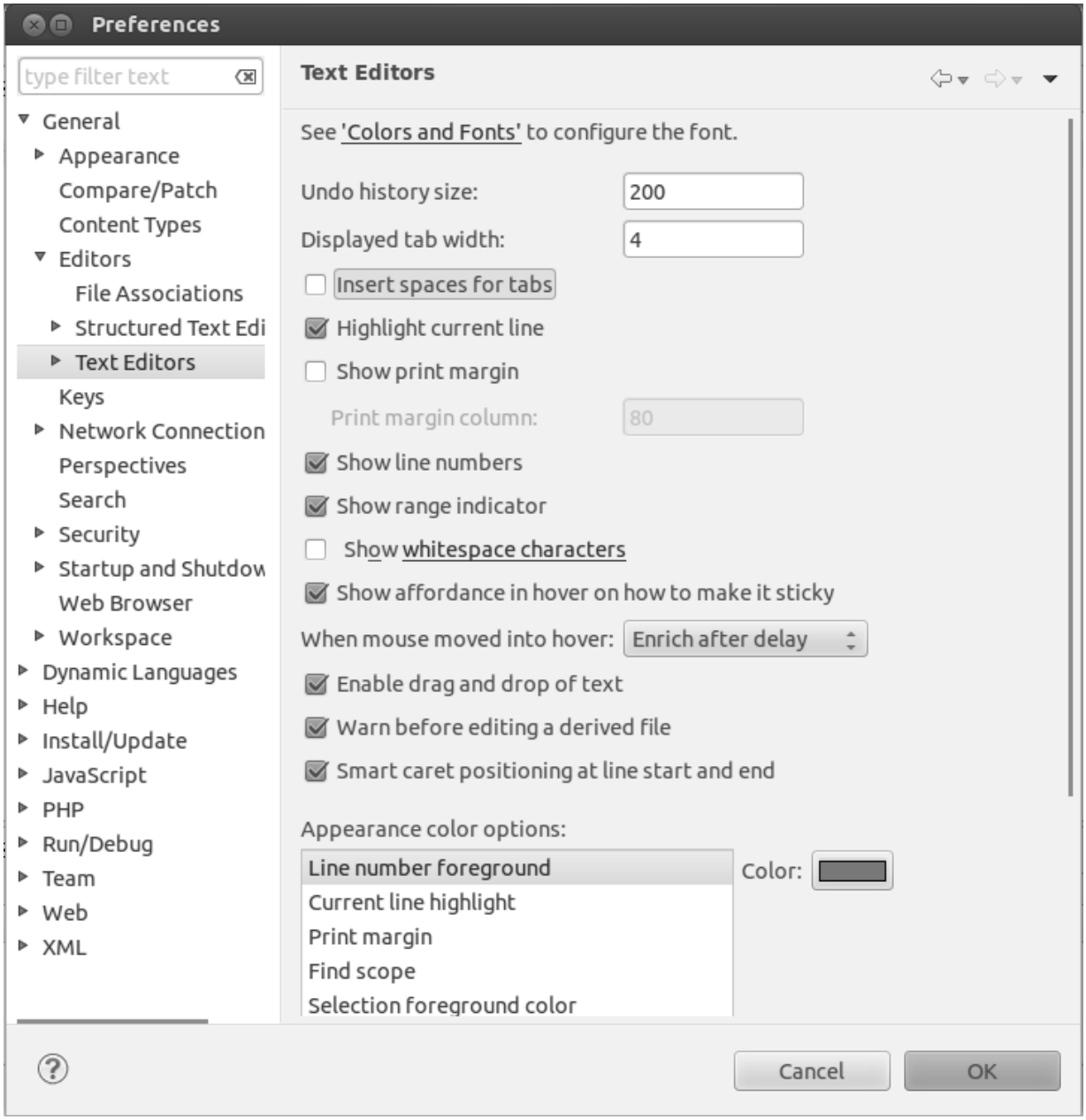


Figura 3.8: Quadro Text Editors

O Eclipse completa automaticamente código PHP, mas você vai perceber que alguns templates de código não estão aderentes às PSRs. Isso é facilmente resolvido editando os templates pelo menu `Window ? Preferences`. Acesse o item `PHP ? Editor ? Templates` e você terá uma janela como a da figura a seguir.

Inclusive, nessa figura, está selecionado o template de classe PHP, onde vemos a abertura de chaves na mesma linha do nome da classe, o que contraria a PSR-2. Para alterar, basta clicar no botão `Edit`, modificar o código, clicar em `OK` e depois em `Apply`. Além de alterar os templates existentes, é possível criar os seus próprios templates. Não vamos criar nenhum, estamos apenas mostrando como você pode fazer se em algum momento precisar.

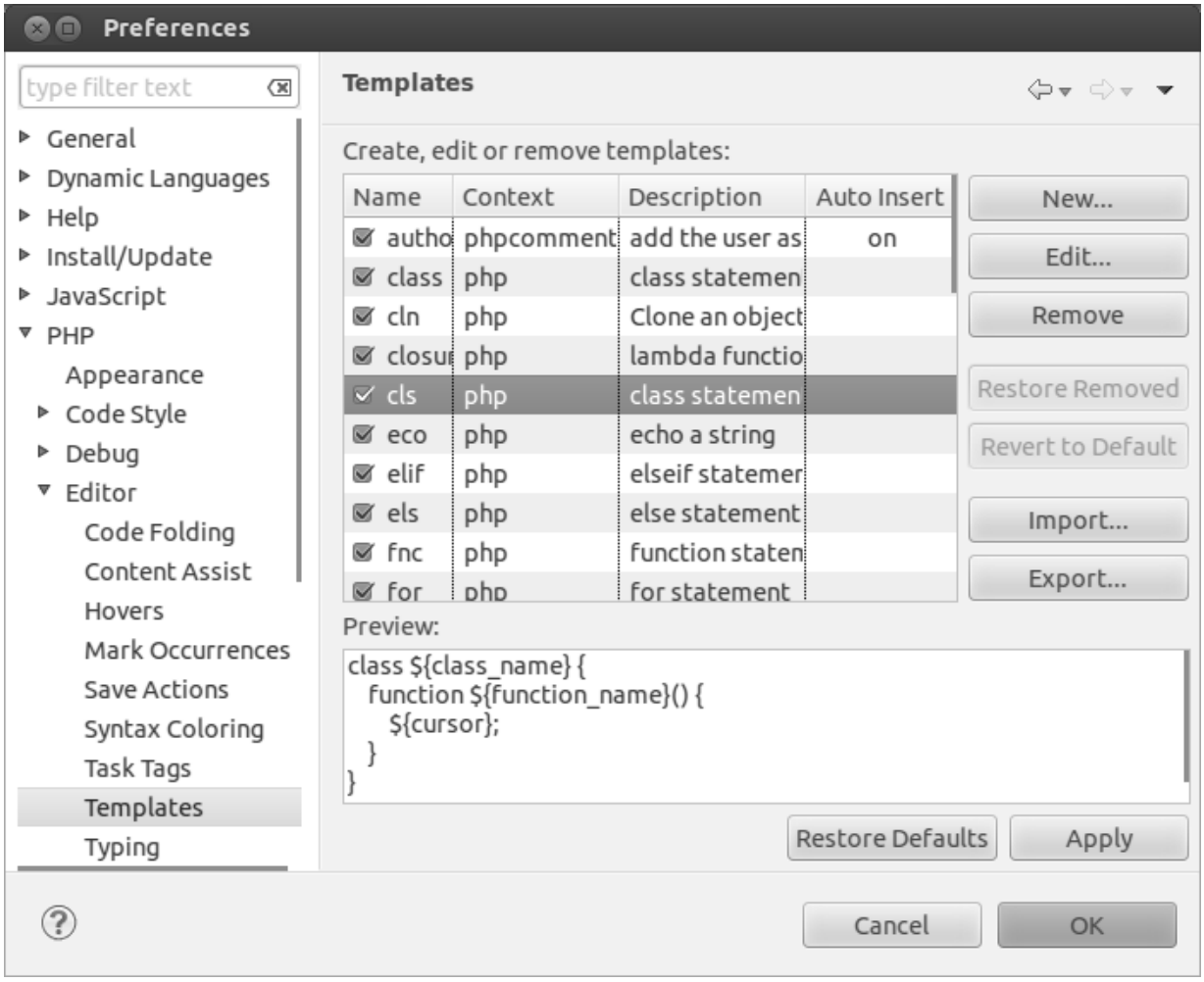


Figura 3.9: Quadro templates

Pronto, já temos as ferramentas instaladas e configuramos nosso ambiente. Estamos prontos para começar nossa aplicação. No próximo capítulo, faremos uma revisão da linguagem de programação PHP.

Capítulo 4
Bússola da estrutura de PHP

O duro e rijo quebra. O flexível resiste. – Tao Te Ching

Em 8 de junho de 1995, o dinamarquês da Groenlândia Rasmus Lerdorf anunciou a primeira versão em código aberto da sua ferramenta para criar páginas web pessoais. Era o Personal Home Page Tools, ou simplesmente PHP Tools. Foi a primeira tecnologia de desenvolvimento dirigida para a World Wide Web. Era simples de usar e podia ser embutida em meio a código HTML.

Segundo Deitel e Deitel (2003, p. 929), “PHP é uma tecnologia de código-fonte aberto que é suportada por uma grande comunidade de usuários e desenvolvedores”. Esta é uma das principais características PHP, embora não seja a mais alardeada.

Por ser um software de código-fonte aberto e livre, os desenvolvedores têm acesso ao seu código-fonte, sabem como ele funciona, podem alterá-lo e redistribuí-lo livremente. Além disso, é uma plataforma independente, o que significa que existem implementações para os sistemas operacionais mais usados no mundo, e tem suporte para a maioria dos sistemas gerenciadores de bancos de dados do planeta.

Conforme Deitel e Deitel afirmam, “o poder da web reside não apenas em servir conteúdo para os usuários, mas também em responder aos pedidos dos usuários e em gerar páginas web com conteúdo dinâmico”. E, de acordo com eles, “enquanto outras linguagens também podem realizar essa função, o PHP foi escrito especificamente para interagir com a web”.

Segundo pesquisa da W3TECHS (2017), PHP é utilizado por mais de 82% de todos os websites cuja linguagem de programação do lado servidor é conhecida. A empresa Netcraft (IDE, 2013)

constatou que o PHP está presente em mais de 200 milhões de websites no mundo.

Este capítulo é o seu guia de referência interno para as características estruturais de PHP válidas a partir da sua versão 5.5.

4.1 Configuração do PHP

O comportamento do PHP pode ser configurado em vários níveis:

Configuração global

O `php.ini` é o arquivo de configuração do PHP, contendo mais de 200 diretivas capazes de alterar praticamente cada aspecto do comportamento da linguagem. Esse arquivo é interpretado cada vez que o PHP é invocado, o que, para a versão módulo do servidor, ocorre somente quando o servidor web inicia, e a cada requisição para a versão CGI (*Common Gateway Interface*).

O protocolo HTTP trabalha apenas com conteúdo estático. O CGI é uma tecnologia que permite adicionar a aplicações que trabalham com o protocolo HTTP a capacidade de geração de conteúdo dinâmico a partir da interação de páginas HTML com scripts executados no servidor.

Configuração específica de diretório e host

Se você não tiver acesso ao arquivo `php.ini`, pode ser capaz de mudar diretivas do PHP de dentro dos arquivos `httpd.conf` ou `.htaccess` do Apache. Por exemplo, para forçar a exibição de todos os erros do PHP somente para seu domínio de desenvolvimento (como: http://dev.fgsl.eti.br), adicione o seguinte código a um arquivo `.htaccess`:

```
php_flag display_errors on
```

Cada diretiva é associada a um dos três níveis de permissão (`PHP_INI_ALL` , `PHP_INI_PER_DIR` , `PHP_INI_SYSTEM`) que determinam onde ele pode ser configurado. Consulte a documentação do PHP antes de sair alterando configurações fora do arquivo `php.ini` . Para uma lista completa de diretivas, acesse http://www.php.net/ini.

Configuração específica de script

Ocasionalmente, você pode querer alterar diretivas em um script PHP que serve de base para uma aplicação. Por exemplo, para alterar o tempo máximo de execução permitido ao PHP para um script encarregado de carregar arquivos grandes para um servidor, você pode chamar a função `ini_set()` de dentro de seu script PHP, desta forma:

```
ini_set('max_execution_time',60);
```

Alterando a extensão de arquivo do PHP

A extensão padrão do arquivo PHP é `.php` , entretanto, você pode alterá-la para qualquer coisa que o deixe feliz; basta adicionar a extensão à diretiva `AddType` dentro do arquivo `httpd.conf` do Apache. Por exemplo, para configurar Apache para reconhecer `.fgsl` como uma extensão de arquivo suportada pelo PHP:

```
AddType application/x-httpd-php .php .fgsl
```

4.2 Tipos de dados

PHP tem tipos de dados escalares, tipos de dados compostos e tipos de dados especiais. Ele é uma linguagem de tipagem fraca, mas não interprete isso erroneamente como uma deficiência, e sim como uma estratégia. As variáveis no PHP não são declaradas com um específico tipo de dados. Em vez disso, o tipo de uma variável é determinado com base no contexto no qual ela é usada. Essa

estratégia é frequentemente referida como logro ou prestidigitação de tipo. É possível usar conversão explícita de tipo, de modo a forçar uma variável a ser avaliada como um tipo específico, como mostra o exemplo a seguir.

```
$text = '1418';
// $text é criada com o tipo string pois recebeu um valor string
$number = (int) $text;
// $number contém o número inteiro 1418
```

A conversão explícita de tipo é feita ao anteceder o valor a ser alterado com o nome do tipo entre parênteses. Você pode consultar http://php.net/types.type-juggling para obter mais detalhes sobre logro de tipo e conversão de tipo.

O PHP tem dois conjuntos de operadores de comparação, que comparam tanto valores quanto tipos, como também valores de acordo com a prestidigitação de tipo.

Operador	Nome	Comparação
==	Igual	Compara o valor de duas variáveis, convertendo-os para um tipo comum. Retorna verdadeiro se os valores resultantes forem iguais.
===	Idêntico	Compara o valor e o tipo de duas variáveis. Não faz conversão e só retorna verdadeiro se tanto o valor quanto o tipo forem iguais.
!=	Diferente	Retorna verdadeiro quando igual retorna falso.
<>	Diferente	Retorna verdadeiro quando igual retorna falso.
!==	Não idêntico	Retorna verdadeiro quando idêntico retorna falso.

Os tipos escalares do PHP são: boolean, integer, float e string. Por definição, tipos de dados escalares contêm um valor único. A tabela a seguir lista funções que analisam o tipo de uma variável.

Função	O que faz
`gettype(var)`	Retorna uma string com o tipo da `var`.
`empty(var)`	Retorna `FALSE` se `var` não for vazia e o valor for diferente de zero.
`is_null(var)`	Retorna `TRUE` se `var` é `null`; caso contrário, `FALSE`.
`isset(var)`	Retorna `TRUE` se `var` existe; caso contrário, `FALSE`.

Boolean

Valores boolean podem ser `true` ou `false`. Quando outro tipo de dados é avaliado em um contexto boolean, como na condição de prosseguimento das estruturas `for` e `while`, os seguintes valores são convertidos implicitamente para `false`:

- O integer 0 (zero);
- O float 0.0 (zero);
- A string vazia e a string `0`;
- O array sem elementos;
- O tipo especial `NULL` (incluindo variáveis não definidas);
- O objeto `SimpleXML` criado a partir de tags vazias;
- E os demais valores são convertidos para `true`.

Integer

Integers são números inteiros que também incluem os negativos (PHP não suporta números inteiros sem sinal). O tamanho de um integer depende da plataforma sobre a qual o PHP está rodando. Literais integer (gerados por atribuição explícita) podem ser

especificados no formato decimal, octal, hexadecimal ou binário (com a função `bindec()`).

```
$n1 = 0123; // número octal (equivalente a 83 em decimal)
$n2 = 0x1A; // número hexadecimal (equivalente a 26 em decimal)
$n3 = bindec('110011'); // número binário (equivalente a 51 em decimal)
```

Quando você atribui um número inteiro com ou sem sinal a uma variável sem usar o molde de tipo, o PHP assume por padrão que é um integer.

Float

Um float é um número que pode conter uma fração. Como integers, o tamanho do float é dependente da plataforma sobre a qual o PHP está rodando. Literais float podem ser especificados no formato decimal ou em notação científica.

```
$n1 = 45; // integer
$n2 = (float) 45; // float
$n3 = 3.1415926; // float
$n4 = 2.6E10; // 2.6 X 10 ^ 10 = 26000000000
$n5 = 2.6E-1; // 2.6 X 10 ^ -1 = 2.6 X (1/10) = 0.26
```

String

Uma string em PHP é tão somente um array de bytes, com cada byte tipicamente representando um caractere. Veremos mais detalhes sobre strings na seção correspondente.

Tipos compostos

Os tipos de dados compostos em PHP são `array` e `object`. Veremos mais detalhes sobre eles respectivamente nas seções correspondentes.

Tipos especiais

Os tipos de dados especiais no PHP são `resource`, `NULL` e `callable`. Um `resource` é um tipo especial que mantém uma referência para algum recurso externo. Esse recurso externo pode ser desde um stream até uma conexão de banco de dados. A lista completa de tipos `resource` está disponível em http://php.net/resource.

NULL é tanto um tipo de dado quanto o único valor possível para esse tipo. Uma variável com valor `NULL` é uma variável com nenhum valor. Ela pode se tornar `NUL` L ao ser associada com o valor especial `NULL`; ao ser declarada, mas ainda não ter nenhum valor configurado; ou por ter seu valor eliminado pela função `unset()`.

Algumas das funções embutidas no PHP aceitam callbacks definidos pelo usuário. Callbacks são variáveis que representam elementos executáveis, como funções e métodos. Uma função pode ser representada por um tipo `string`, contendo o nome da função, ou por um objeto da classe `Closure`.

Um método é representado por um array, contendo dois elementos: um tipo `string` com o nome da classe, se o método for estático; ou um objeto e um tipo `string` com o nome do método. Esses parâmetros são considerados do tipo `callable`. Esse tipo pode também ser usado como um tipo sugestivo dentro de funções e métodos definidos pelo usuário.

Variáveis variáveis (ponteiros para variáveis)

PHP permite que você utilize uma variável para se referir a um nome de variável de forma indireta. No exemplo a seguir, usamos a variável `$a` para nos referirmos à variável `$b`.

```
$a = 'b';
$b = 4;
echo $$a; // imprime 4, que é o valor de $b
```

Isso é chamado de *variável variável*. É uma variável que aponta para outra variável. Você pode fazer encadeamento de variáveis variáveis, como no exemplo adiante.

```php
$a = 'b';
$b = 'c';
$c = 7;
echo $$$a; // imprime 7, que é o valor de $c
```

Cópia e referência

Variáveis são manipuladas por cópia ou referência. Quando atribuímos uma variável à outra, a variável receptora passa a ter uma cópia do valor da doadora e torna-se independente dela. O trecho a seguir mostra como funciona a cópia de valores.

```php
$a = 1;
$b = $a;
$a++;
echo $b; // imprimirá 1, mesmo que $a tenha o valor 2 agora
```

O operador `&` permite que façamos referência ao endereço da variável, e assim podemos ter mais de uma variável que aponte para o mesmo endereço, como no exemplo a seguir.

```php
$a = 1;
$b = &$a;
$a++;
echo $b; // imprimirá 2, pois $b aponta para $a
```

Impressão de variáveis

Você pode imprimir variáveis com o comando `echo` ou a função `print_r()`. O comando `echo` imprime apenas strings, de modo que, se você tentar imprimir um array ou um objeto, vai provocar um erro. A função `print_r()`, por outro lado, imprime qualquer tipo de dado. Ela pode enviar seu retorno diretamente para a saída padrão, ou atuar como uma função comum se o segundo argumento opcional receber o valor `true`.

4.3 Strings

Sintaxe de strings

PHP suporta diversas sintaxes para expressar strings: apóstrofos, aspas, *heredocs* e *nowdocs*. Strings delimitadas por apóstrofos constituem a mais simples sintaxe, pois não há processamento sobre o conteúdo, exceto para a barra invertida.

Para imprimir um apóstrofo, use `\'` , pois a barra invertida neutraliza-o. Para imprimir uma barra invertida e um apóstrofo, use `\\\'` , já que a barra invertida é neutralizada por ela mesma.

```
$numero = 250;
$texto = 'Geraldo mora na casa de número $numero';
echo $texto; // imprimirá 'Geraldo mora na casa de número $numero';
```

Strings delimitadas por aspas suportam interpolação de variáveis. Isso significa que você pode colocar uma variável dentro de uma string delimitada por aspas, assim, o valor da variável será expandido na string resultante.

Além disso, as aspas permitem incluir uma série de sequências para caracteres especiais, como `\n` para final de linha e `\t` para tabulação horizontal. Para imprimir aspas duplas, use `\"` . Para imprimir o cifrão (`$`), use `\$` .

```
$numero = 250;
$texto = "Geraldo mora na casa de número $numero";
echo $texto; // imprimirá 'Geraldo mora na casa de número 250';
```

A sintaxe *heredocs* trabalha de modo similar às strings delimitadas por aspas. Ela facilita a criação de textos em múltiplas linhas e também suporta interpolação de variáveis. Essa sintaxe também evita a criação de textos longos pela concatenação de strings e deixa claro como ficará o resultado.

Por exemplo, para criar um parágrafo de três linhas, podemos usar o operador de concatenação (`.`) e as sequências `\n` e `\t` , conforme o trecho de código a seguir.

```php
$nome = 'Pato Donald';
$numero = 1313;
$texto = "\tO digníssimo senhor $nome \n";
$texto .= "reside na bela e arborizada rua das patacas\n";
$texto .= "no número $numero ao lado de uma enorme mangueira.";
echo $texto;
/**
O texto impresso será exatamente assim:
    O digníssimo senhor Pato Donald
reside na bela e arborizada rua das patacas
no número 1313 ao lado de uma enorme mangueira.

 */
```

Com *heredocs*, o texto pode ser montado de uma forma mais simples e legível, conforme mostra o exemplo:

```php
$nome = 'Pato Donald';
$numero = 1313;
$texto = <<<TEXT
    O digníssimo senhor $nome
reside na bela e arborizada rua das patacas
no número $numero ao lado de uma enorme mangueira.
TEXT;
echo $texto;
/**
O texto impresso será exatamente assim:
    O digníssimo senhor Pato Donald
reside na bela e arborizada rua das patacas
no número 1313 ao lado de uma enorme mangueira.

 */
```

A palavra `TEXT` é um identificador de delimitação do texto. O identificador deve ser usado na abertura, precedido por `<<<` e, no fechamento, sucedido por ponto e vírgula. O fechamento sempre tem de ser feito na primeira coluna, e não é permitida indentação nesse caso.

O `TEXT` é um exemplo, poderia ser ASTERIX, CEBOLINHA, TINTIN ou qualquer outro nome, desde que em letras maiúsculas. A barra invertida pode ser usada em *heredocs* para neutralizar o cifrão (`$`), de modo a exibir o nome de uma variável, e não seu conteúdo.

Nowdocs são para *heredocs* o que os apóstrofos são para as aspas. *Nowdocs* não permitem interpolação de variáveis. A diferença consiste em cercear o identificador de delimitação com apóstrofos. O exemplo a seguir mostra como usar *nowdocs* para imprimir os nomes de variáveis no lugar de seus valores.

```
$nome = 'Pato Donald';
$numero = 1313;

$texto = <<<'TEXT'
    O digníssimo senhor $nome
reside na bela e arborizada rua das patacas
no número $numero ao lado de uma enorme mangueira.
TEXT;

echo $texto;

/**
O texto impresso será exatamente assim:

    O digníssimo senhor $nome
reside na bela e arborizada rua das patacas
no número $numero ao lado de uma enorme mangueira.

 */
```

Manipulação de strings

O PHP suporta mais de cem funções identificadas como específicas para tratamento e manipulação de strings.

Exemplos de manipulação de strings

- Converter um array para uma string:

```php
$channels = array('Disney','Discovery','Nicklodeon');
$channels = implode(',',$channels);
// O conteúdo final de $channels é a string 'Disney,Discovery,Nicklodeon'
```

- Converter uma string para um array:

```php
$channels = 'Disney,Discovery,Nicklodeon';
$channels = explode(',',$channels);
// $channels[0] = 'Disney', $channels[1]='Discovery',
$channels[2]='Nicklodeon';
```

- Contar palavras em uma string:

```php
$sentence = 'Não atirem antes de ver o branco dos olhos deles';
$words = str_word_count($sentence);
// O conteúdo de $words é o inteiro 10
// Veja também: count_chars()
```

- Converter uma string para letras maiúsculas:

```php
$avenger = strtoupper('hulk');
// o conteúdo de $avenger é a string 'HULK'
// Veja também: lcwords(), strtolower(), ucfirst(), ucwords()
```

- Extrair tags HTML e PHP de uma string:

```php
$input = 'Você ganhou na <a href="http://www.exemplo.com.br">loteria!
</a>';
$clean = strip_tags($input);

// O conteúdo de $clean é a string 'Você ganhou na loteria!'
// Veja também: htmlentities(), htmlspecialchars()
```

- Substituir todas as ocorrências de uma substring:

```php
$phrase = 'O Palmeiras é o campeão dos campeões';
$phrase = str_replace('Palmeiras','Corinthians',$phrase);
// O conteúdo final de $phrase é a string 'O Corinthians é o campeão dos
campeões'
// Veja também: substr_replace(), strireplace(),strtr()
```

- Extrair parte de uma string:

```php
$description = 'A música que o povo gosta';
$begin = substr($description,2,7);
// O conteúdo de $begin é a string 'música'
// Veja também: strrchr()
```

- Comparar duas strings de modo *case-insensitive*:

```php
if (strcasecmp('Marco Aurélio','marco aurélio') == 0)
echo 'As strings são iguais em um contexto case-insensitive.';
// Veja também: strncasecmp()
```

- Converter caracteres de novas linhas para a tag HTML `<br />` :

```php
$members = "Lashorr: 3453\n Gpaak: 3515\n Harvid: 2937\n";
$html = nl2br($members);
// O conteúdo de $html é a string 'Lashorr: 3453<br /> Gpaak: 3515<br />
Harvid: 2937<br />'
// Veja também : htmlentities(), htmlspecialchars()
```

Funções gerais para strings

Função	Descrição
`bin2hex`	Retorna a representação hexadecimal dos caracteres de uma string.
`chr`	Retorna um caractere pelo seu código ASCII.
`crypt`	Criptografa uma string usando o algoritmo e salt especificados.
`hex2bin`	Retorna caracteres pela sua representação hexadecimal.
`lcfirst`	Retorna uma string com o primeiro caractere minúsculo.
`levenshtein`	Calcula a distância Levenshtein entre duas strings.
`ltrim`	Retorna uma string sem caracteres em branco (ou outros) à esquerda.

Função	Descrição
`money_format`	Retorna um número como uma moeda.
`ord`	Retorna o código ASCII de um caractere (o inverso de `chr`).
`parse_str`	Converte uma string em variáveis.
`similar_text`	Calcula a similaridade entre dois textos.
`soundex`	Retorna uma chave representando a pronúncia de um texto.
`str_getcsv`	Retorna um array a partir de um texto no formato CSV.
`str_ireplace`	Faz o mesmo que `str_replace`, mas ignora minúsculas e maiúsculas.
`str_pad`	Preenche o tamanho almejado de uma string com uma substring.
`str_repeat`	Replica uma string o número de vezes especificado.
`str_shuffle`	Mistura aleatoriamente os caracteres de uma string.
`str_split`	Transforma uma string em um array.
`stripos`	Encontra a primeira ocorrência de uma substring.
`strpos`	O mesmo que `stripos`, mas considerando minúsculas e maiúsculas.
`substr_count`	Conta o número de ocorrências de uma substring.
`substr_replace`	Substitui as ocorrências de um texto em uma substring.

Função	Descrição
`trim`	Retira espaços em branco à esquerda e à direita da string.
`ucfirst`	Retorna uma string com o primeiro caractere maiúsculo.

Funções de formatação de entrada e saída de texto

Função	Descrição
`fprintf`	Grava uma string formatada para um stream.
`print`	Envia uma string para a saída padrão.
`printf`	Mostra uma string formatada.
`sprintf`	Retorna uma string formatada.
`sscanf`	Interpreta um valor de entrada de acordo com um formato.
`vfprintf`	Grava uma string formatada para um stream.
`vprintf`	Mostra uma string formatada.
`vsprintf`	Retorna uma string formatada.

As funções que têm o mesmo nome de outras com uma letra `v` prefixando-as retornam o mesmo resultado, mas aceitam um de seus argumentos como um array no lugar de um número variável de argumentos.

Expressões regulares

As expressões regulares em PHP emprestam as fortes características tanto do padrão Perl quanto do POSIX, e na verdade

são formalmente identificadas como tais.

Funções de expressão regular compatíveis com Perl (PCRE)

PHP suporta oito funções específicas PCRE, incluindo estas soluções mais utilizadas:

Função	O que faz
`preg_grep`	Procura `$subject` para `$pattern`, retornando um array de casamentos. O parâmetro opcional `$flags` pode ser configurado para `PREG_GREP_INVERT`, gerando um array consistindo de elementos não casados a serem retornados.
Assinatura	`array preg_grep(str $pattern, array $subject [, int $flags])`
`preg_match`	Determina se `$pattern` existe em `$subject`. Se `$matches` for definido, uma variável nomeada similarmente será retornada contendo os casamentos. Se `$flags` for configurada para `PREG_OFFSET_CAPTURE`, o valor da string `$offset` será retornada para cada casamento. Veja `preg_match_all()` para uma variação dessa função.
Assinatura	`int preg_match(str $pattern, str $subject [, array &matches [, int $flags [, int $offset]]])`
`preg_replace`	Procura `$subject` para `$pattern`, substituindo quaisquer instâncias com `$replacement`. Veja `preg_replace_callback()` para uma variação dessa função.
Assinatura	`mixed preg_replace(mixed $pattern, mixed $replacement, mixed $subject [, int $limit [, int &$count]])`

- Modificadores de padrão PCRE comuns:

Modificador	Descrição

Modificador	Descrição
`g`	Executa uma busca global.
`i`	Executa uma busca case-insensitive.
`m`	Trata a string como múltiplas linhas.
`s`	Ignora os caracteres de novas linhas.
`x`	Ignora espaços em branco e comentários.
`u`	Para no primeiro casamento (busca não ambiciosa).

- Metacaracteres:

Metacaractere	Descrição
`\A`	Casa somente o início da string.
`\b`	Casa uma palavra fronteiriça (no início ou no fim).
`\B`	Casa qualquer palavra que não seja fronteiriça.
`\d`	Casa um caractere representando um dígito.
`\D`	Casa um caractere que não seja um dígito.
`\s`	Casa um caractere de espaço em branco.
`\S`	Casa um caractere que não seja um espaço em branco.
`[]`	Envolve classes de caracteres.
`()`	Envolve um agrupamento de caracteres.
`$`	Casa com o fim da linha.
`^`	Casa o começo da linha.

Metacaractere	Descrição
`.`	Casa qualquer caractere, exceto novas linhas.
`\`	Anula o efeito do próximo metacaractere.
`\w`	Casa qualquer string contendo caracteres underscore (_) e alfanuméricos.
`\W`	Casa uma string contendo qualquer coisa exceto underscore e caracteres alfanuméricos.

Funções de expressão regular POSIX

O PHP suporta sete funções, como definido pela especificação POSIX 1003.2. No entanto, saiba que as expressões regulares POSIX não são seguras para dados binários. As funções POSIX mais frequentemente usadas são:

Função	O que faz
`ereg`	Procura `$string` para um `$pattern`. Você pode opcionalmente incluir o parâmetro `$regs`, que vai gerar um array com o mesmo nome, a ser retornado contendo cada casamento. Veja `eregi()` para a contrapartida case-insensitive.
Assinatura	`int ereg(str $pattern, str $string [, array &$regs])`
`ereg_replace`	Substitua qualquer `$pattern` encontrado em `$string` com `$replacement`. Veja `eregi_replace()` para a contrapartida case-insensitive.
Assinatura	`string ereg_replace(str $replace, str $string [, int $limit])`
`split`	Coloca o conteúdo de `$string` dentro de um array, dividindo-o de acordo com `$pattern`. Veja `spliti()` para a contrapartida case-insensitive.

Função	O que faz
Assinatura	`array split(str $pattern, str $string [, int $limit])`

Sintaxe de expressão regular POSIX

Código	Descrição
[0-9]	Qualquer dígito decimal de 0-9.
[a-z]	Qualquer caractere de `a` minúsculo a `z` minúsculo.
[A-Z]	Qualquer caractere de `A` maiúsculo a `Z` maiúsculo.
[A-Za-z]	Qualquer caractere de `A` maiúsculo a `z` minúsculo.
p+	Qualquer string contendo ao menos um `p`.
p*	Qualquer string contendo zero ou mais ocorrências de `p`.
p?	Qualquer string contendo zero ou um `p`.
p{N}	Qualquer string contendo sequências de `N p`.
p{N,M}	Qualquer string contendo sequências entre `N` e `M p`.
p{2,}	Qualquer string contendo sequências de pelo menos `2 p`.
p$	Qualquer string contendo `p` no final.
^p	Qualquer string contendo `p` no início.
a-zA-Z	Qualquer string não contendo caracteres entre `a-z` e `A-Z`.

Código	Descrição
p.p	Qualquer string contendo `p` seguido por qualquer caractere, seguido por outro `p`.

Exemplos de expressões regulares

- Validando um número de telefone (presuma que o formato requerido é `xxx-xxx-xxxx`):

```
// PCRE
if (preg_match('/^[2-9]{1}\d{2}-\d{3}-\d{4}$/','313-616-1313'))
    echo 'Número válido!';
// POSIX
if (ereg_match('^[2-9]{1}[0-9]{2}-[0-9]{3}-[0-9]{4}$','313-616-1313'))
    echo 'Número válido!';
```

- Validando um nome de usuário (presuma que o nome de usuário esteja entre 6 e 10 caracteres numéricos e alfabéticos):

```
// PCRE
if (preg_match('/^[a-z0-9]{6,10}$/i', '800bozo'))
echo "Nome de usuário válido!";
// POSIX
if (eregi('^[a-z0-9]{6,10}$', '800bozo'))
echo "Nome de usuário válido!";
```

- Transformando URLs em hyperlinks:

```
// PCRE
$text = "Vá para http://www.bozo.com.";
$html = preg_replace('/\s(\w+:\/\/)(\S+\.?)(\w+)/',
' <a href="\\1\\2\\3">\\1\\2\\3</a>', $text);
// POSIX
$text = "Vá para http://www.bozo.com. ";
$html= ereg_replace('[a-zA-Z]+://(([.]?[a-zA-Z0-9_/-])*)',
'<a href="\\0">\\0</a>', $string);
// $html = "Vá para <a href=" http://www.bozo.com">http://www.bozo.com."
```

4.4 Arrays

Arrays PHP são mapas ordenados, o que significa que valores são mapeados para chaves. Arrays podem ser usados como arrays, listas, tabelas hash, dicionários, pilhas e filas. Embora arrays PHP suportem muitas estruturas de dados diferentes, a *Standard PHP Library* (SPL) provê estruturas de dados especializadas, que são geralmente mais rápidas e usam menos memória (mas são menos flexíveis). Para conhecer as estruturas disponíveis pela SPL, consulte http://php.net/spl.datastructures.

PHP suporta tanto arrays enumerativos quanto associativos. Os enumerativos são indexados por uma chave do tipo integer, enquanto os associativos são indexados por uma chave do tipo string. Valores de um array podem ser de qualquer tipo de dados do PHP, incluindo arrays. Quando um array contém outro array, ele é referido como um array multidimensional.

Criando um array

O exemplo seguinte criará o array `$nephews`, consistindo de três elementos, com o uso da função `array()`:

```
$nephews = array(
        'Dewey',
        'Joey',
        'Louis'
);
```

O mesmo array pode ser criado de forma implícita com o uso de colchetes:

```
$nephews = [
        'Dewey',
        'Joey',
        'Louis'
];
```

Você pode criar um array, com um intervalo baseado em caracteres ou números, usando a função `range()` :

```
// $problemAge = array(13,14,15,16,17,18,19); ou $problemAge =
[13,14,15,16,17,18,19]
$problemAge = range(13,19);
```

Recuperando conteúdos de um array

Arrays indexados, tais como os criados anteriormente, podem ser acessados de acordo com o seu intervalo numérico. Por exemplo, para recuperar o segundo valor no array `$nephews` :

```
$blueCap = $nephews[1];
```

Talvez o modo mais flexível para enumerar o conteúdo de um array seja por meio da declaração `foreach` :

```
foreach ($nephews as $nephew) {
    printf("%s<br />",$nephew);
}
```

Arrays associativos

Arrays associativos dão aos desenvolvedores a oportunidade de associar contexto significativo, tanto para o valor do array quanto para sua chave correspondente:

```
$nephews = array(
        'Dewey' => 'red cap',
        'Joey' => 'blue cap',
        'Louis' => 'green cap'
);
```

Você pode obter o valor (neste caso, a cor do boné) pela referência ao nome do sobrinho:

```
$cap = $nephews['Louis'];
```

A declaração `foreach` prova-se igualmente útil para navegar arrays associativos:

```
foreach($nephews as $name => $cap) {
    printf("%s => %s<br />", $name, $cap);
}
```

Arrays multidimensionais

Arrays multidimensionais são úteis para representar estruturas de dados mais complexas:

```
$heroGroups = array(
        'Avengers' => array(
                'Iron Man' => 'Tony Stark',
                'Thor' => 'Donald Blake',
                'Scarlet Witch' => 'Vanda Maximoff',
                'Captain America'=> 'Steve Rogers',
                'Black Panther'=>'T\'Challa',
        ),
        'Defenders' =>    array(
                'Doctor Strange' => 'Stephen Strange',
                'Submariner' => 'Namor',
                'Silver Surfer' => 'Norrin Radd',
                'Hulk'=>'Bruce Banner',
                'Hawkeye'=>'Clint Barton'
        )
);
```

Referenciar um elemento não é diferente dos métodos usados para arrays indexados e associativos; é apenas um pouco mais extenso:

```
$medician = $heroGroups['Defenders']['Doctor Strange'];
```

Determinando o tamanho de um array

O número de elementos encontrados em um array pode ser determinado usando a função `count()` :

```
printf("catalogamos %d grupos de heróis",count($heroGroups));
// Imprime "catalogamos 2 grupos de heróis"
```

Ordenando arrays

O PHP oferece um poderoso arsenal de funções capazes de ordenar arrays de uma variedade de modos. A maioria dessas funções aceita um parâmetro opcional que pode mudar o comportamento de ordenação.

Quatro valores são suportados, incluindo: `SORT_REGULAR` , para comparar elementos sem a moldagem de tipo implícita; `SORT_NUMERIC` , para comparar elementos numericamente; `SORT_STRING` , para comparar elementos como strings; e `SORT_LOCALE_STRING` , para ordenar elementos de acordo com a localização definida.

Função	O que faz
`asort`	Ordena um array mantendo a associação de chave.
Assinatura	`bool asort(array &array [, int $sort_flags])`
`arsort`	Ordena um array associativo de forma reversa, mantendo a associação de chave.
Assinatura	`bool arsort(array &array [, int $sort_flags])`
`ksort`	Ordena um array associativo pela chave, mantendo a associação de índice.
Assinatura	`bool ksort(array &array [, int $sort_flags])`
`krsort`	Ordena um array associativo de forma reversa por chave, mantendo a associação de índice.
Assinatura	`bool krsort(array &array [, int $sort_flags])`
`natcasesort`	Ordena um array de modo case-insensitive na ordem logicamente presumida por humanos.
Assinatura	`bool natcasesort($array &array)`
`natsort`	Ordena um array na ordem logicamente presumida por humanos.

Função	O que faz
Assinatura	`bool natsort($array &array)`
`rsort`	Ordena um array de forma reversa.
Assinatura	`bool rsort(array &array [, int $sort_flags])`
`usort`	Ordena um array de acordo com as especificações de uma função definida pelo usuário.
Assinatura	`bool usort(array &array, callback$comparison_function)`
`uasort`	Ordena um array de acordo com as especificações de uma função definida pelo usuário, mantendo a associação de índice.
Assinatura	`bool uasort(array &array, callback $comparison_function)`
`uksort`	Ordena as chaves de um array de acordo com as especificações de uma função definida pelo usuário.
Assinatura	`bool uksort(array &array, callback $comparison_function)`

Gerando coleções a partir de arrays

Algumas vezes é necessário gerar um array a partir de uma iteração com dados de outro array ou `Iterator`. A palavra-chave `yield` permite gerar um array e seus elementos de forma implícita, evitando a necessidade de criar uma variável do tipo array e o seu retorno. Veja o seguinte código, por exemplo:

```
$options = array();
foreach($records as $record){
    $options[] = $record->name;
}
return $options;
```

Ele pode ser reescrito da seguinte forma:

```
foreach($records as $record){
    yield $record->name;
}
```

O comando `yield` cria um objeto da classe `Generator`, que implementa a interface `Iterator`. O retorno desse objeto ao final de uma função ou método é implícito.

O comando `yield` também aceita a passagem da chave do array, permitindo a criação de arrays associativos na coleção gerada. Por exemplo, o código a seguir:

```
$options = array();
foreach($records as $record){
    $options[$record->codigo] = $record->name;
}
return $options;
```

Ele pode ser reescrito da seguinte forma:

```
foreach($records as $record){
    yield $record->codigo => $record->name;
}
```

Aqui concluímos os fundamentos da estrutura da linguagem de programação PHP, com seus tipos de dados e as funções diretamente relacionadas à manipulação desses tipos. No próximo capítulo, abordaremos a construção de funções personalizadas e a implementação da programação orientada a objetos.

Capítulo 5
Bússola de funções e classes de PHP

Eu sou mansa, mas minha função de viver é feroz. – Clarice Lispector

Cerca de dois anos após a criação de PHP, os estudantes Zeev Suraski e Andi Gutmans começaram a trabalhar na reescrita do seu mecanismo central de scripting. A partir daí, PHP iniciou um crescimento exponencial e um amadurecimento baseado em três pilares: performance, robustez e segurança. Seu acrônimo mudou de significado para o recursivo PHP Hypertext Preprocessor.

PHP possui uma poderosa coleção de bibliotecas de funções para os mais variados propósitos. E graças à intervenção de Suraski e Gutmans, existem dezenas de bibliotecas de classes disponíveis que ajudam a reaproveitar funcionalidades complexas.

Este capítulo é o seu guia de referência interno para criação e uso funções e classes em PHP.

5.1 Funções

PHP tem mais de mil funções internas ou embutidas. O número de funções internas depende de quais extensões estão instaladas, sendo que ele tem mais de vinte categorias de extensões; cada uma contém de duas a cerca de trinta extensões. Em adição às funções internas, PHP suporta funções definidas pelo usuário. Funções podem aceitar argumentos e retornar valores.

Em programação, há dois conceitos de sub-rotina: os procedimentos e as funções. A diferença é o retorno de resultados: os procedimentos não retornam nada. Algumas linguagens de

programação possuem estruturas sintáticas diferenciadas para os dois conceitos, mas PHP usa a mesma construção governada pela palavra-chave `function`. Uma função PHP pode funcionar como um procedimento quando não usa a instrução `return`.

Criando uma função

O elemento essencial de uma função definida pela pessoa programadora é o nome. Os argumentos são opcionais assim como o valor de retorno. Em virtude da tipagem fraca, argumentos de função podem receber mais de um tipo (desde que não sejam moldados com um tipo de classe ou interface). Por esse mesmo motivo, uma função pode retornar tipos de dados diferentes para chamadas distintas.

Definindo uma função sem argumentos

- Como procedimento (sem retorno de valor):

```
function function_name() {
    // instruções;
}
```

- Como função (com retorno de valor):

```
function function_name() {
    // instruções;
    return $value;
}
```

Definindo uma função com argumentos

Os argumentos em funções são sempre passados por cópia, a menos que seja utilizado o operador `&` em sua declaração. Passagem por cópia significa que a variável passada como argumento não será modificada pela função, pois será gerada uma cópia dela internamente. Na passagem por referência, a variável passada como argumento será modificada pela função. A seguir,

veja um exemplo de função com argumentos com passagem por cópia:

```
function function_name($param1, $param2, $param3) {
    // instruções;
    return $value;
}
```

Agora um exemplo de função com argumentos com passagem por referência:

```
function function_name(&$param1, &$param2, &$param3) {
    // instruções;
    return $value;
}
```

É possível usar argumentos com passagem por cópia e referência na mesma função (desde que sejam argumentos distintos, é claro).

Definindo uma função com argumentos opcionais

```
function function_name($param1, $param2 = null, $param3 = null) {
    // instruções;
    return $value;
}
```

Neste caso, é necessário passar obrigatoriamente apenas `$param1`, desta forma:

```
function_name('qualquer coisa');
```

Porém, para passar o primeiro e o terceiro argumentos, também se faz necessário passar alguma coisa para o segundo, mesmo que seja um `NULL`:

```
function_name('qualquer coisa', NULL, 'outra coisa');
```

Definindo uma função com argumentos tipados

Argumentos de funções podem ser tipados com classes ou interfaces. Se o valor passado não corresponder à classe ou à

interface, será lançada a exceção `TypeError`. Veja:

```
function function_name(ArrayAccess $param1, SplFileInfo $param2,
InfiniteIterator $param3) {
    // instruções;
    return $value;
}
```

Funções variáveis

Uma variável pode conter o nome de uma função e ser utilizada para invocá-la.

```
function afim($a,$x,$b) {
    return $a * $x + $b;
}
$function = 'afim';
echo $function(2,3,5); // é o mesmo que invocar afim(2,3,5)
```

Funções anônimas

PHP tem dois tipos de funções anônimas na forma de lambdas e closures. Funções anônimas precisam terminar com ponto e vírgula, e podem ser referenciadas por variáveis.

Lambdas

A lambda opera de forma similar à função variável. A diferença é que a variável usada para invocar a função não contém uma string com o nome da função, mas, sim, uma referência à definição da função. Veja um exemplo:

```
$afim = function ($a,$x,$b) {
    return $a * $x + $b;
};
echo $afim(2,3,5);
```

Closures

Closures funcionam como lambdas, com a adição do operador `use` que permite que elas utilizem variáveis definidas externamente.

```
$a = 2;
$b = 5;
$afim = function ($x) use ($a,$b) {
    return $a * $x + $b;
};
echo $afim(3);
```

Funções com variáveis estáticas

A palavra-chave `static` assinala que o valor de uma variável deve ser preservado quando a função que a declara for encerrada, de modo que, na próxima chamada, ela não será inicializada. A seguinte função usa uma variável `static` para implementar a operação de potência para expoentes inteiros positivos:

```
function power($base, $expoent) {
    static $power = 1;
    $power *= $base;
    return $expoent == 0 ? 1 : $base * power($base, $expoent-1);
}
```

No exemplo anterior, a variável `$power` é igual a `1` na primeira chamada, mas, a partir da segunda, ela sempre tem o último valor atribuído pela função.

5.2 Classes e objetos

Criando uma classe

Uma classe define o comportamento e as características de uma entidade que você gostaria de representar em uma aplicação. Um exemplo de classe é o seguinte:

```
class Hero {
    private $id;
    private $name;
    private $secretIdentity;
    private $strength;
    private $speed;
    private $vulnerability;

    public function getId() {
        return $this->id;
    }

    public function setId($id) {
        return $this->id = $id;
    }
...
}
```

Instanciando objetos

Para criar uma instância de uma classe (conhecida como objeto), você chama o nome da classe como se fosse uma função, precedendo-o com a palavra-chave `new`:

```
$hero = new Hero();
```

Clonando objetos

O operador `new` sempre retorna a referência para um objeto. Por isso, atribuir a uma variável o valor de outra utilizada para referenciar um objeto tem como efeito compartilhar a referência e não criar uma cópia do objeto.

No exemplo de código a seguir, as variáveis `$kalEl` e `$clarkKent` contêm a referência para a mesma instância da classe `Superman`.

```
$kalEl = new Superman();
$clarkKent = $kalEl;
```

Para criar uma cópia de um objeto, usamos o operador `clone`. No exemplo de código a seguir, as variáveis `$kalEl` e `$clarkKent` contêm referências a instâncias distintas da classe `Superman`.

```
$kalEl = new Superman();
$clarkKent = clone $kalEl;
```

Construtores de classe

Construtores são úteis para executar tarefas de inicialização em tempo de instanciação de classe, evitando o incômodo de chamar métodos de classe adicionais. Construtores são declarados com o uso do método `__construct()`, deste modo:

```
public function _ _construct($id = null) {
    // Se um ID específico é requisitado, recupera-o do banco de dados
    if (!empty($id)) {
        $this->find($id);
    }
}
```

Destrutores de classe

Destrutor não é apenas o irmão do Ciclope dos X-Men. Destrutores de classes customizados podem executar tarefas quando um objeto é destruído. Você pode criar um destrutor usando o método `__destruct()`:

```
private function _ _destruct() {
    printf("O herói %s foi destruído",$this->name);
}
```

Visibilidade de atributo e método

PHP suporta três níveis de visibilidade de atributo e método:

Nível	Descrição

Nível	Descrição
Public	Atributos e métodos públicos podem ser acessados de qualquer lugar.
Private	Atributos e métodos privados são acessíveis somente de dentro da classe que os define.
Protected	Atributos e métodos protegidos estão disponíveis para a classe que os define e para suas subclasses.

Constantes de classe

Constantes de classe são definidas com a palavra-chave `const`, e podem ser referenciadas por meio do operador de resolução de escopo (`::`). Por exemplo, para definir uma constante identificando a versão mínima de PHP suportada pela classe `Hero`:

```
const MIN_PHP_VER = '5.3';
```

Você pode referenciá-la depois da classe, deste modo:

```
echo Hero::MIN_PHP_VER;
```

Atributos estáticos

Atributos pertencem aos objetos por padrão, de modo que cada instância de uma classe governa seus próprios dados. Porém, é possível definir atributos para a classe que sejam visíveis para todas as instâncias e que possam ser manipulados inclusive sem a necessidade de instanciar a classe. São os atributos estáticos, identificados com a palavra-chave `static`.

```
private static $instanceCounter = 0;
```

Atributos estáticos públicos podem ser referenciados por meio do operador de resolução de escopo (`::`):

```
echo Calendar::today;
```

Estendendo classes

Hierarquias de classes podem ser criadas usando a palavra-chave `extends`. Por exemplo, uma aplicação comprometida em catalogar personagens de ficção poderia inicialmente definir uma classe `FictionCharacter`, a qual define algumas características gerais, assim, classes-filha como `Hero` e `Villain` herdariam dela:

```
class FictionCharacter {
    protected $creators;
    protected $creationDate;

    public function setCreators(array $creators) {
        $this->creators = $creators;
    }
    ...
}

class Hero extends FictionCharacter {
    ...
}
```

Se você quiser evitar que classes-filha (neste caso, `FictionCharacter`) sobrescrevam um método-pai, prefixe-as com a palavra-chave final:

```
final public function setCreators(array $creators)     {
    $this->creators = $creators;
}
```

Abstração de classe

A anteriormente mencionada classe `FictionCharacter` seria mais precisamente definida como uma classe abstrata, porque nunca será explicitamente instanciada (em vez disso, seriam instanciadas classes derivadas como `Hero`, `Villain`, `Supporting` etc.).

Classes abstratas são declaradas usando a palavra-chave `abstract`:

```
abstract class FictionCharacter {
    ...
```

```
}
```

Você pode decidir sobrescrever quaisquer métodos encontrados dentro de uma classe abstrata, que seria então herdada por suas classes-filhas. Alternativamente, você também pode declará-las como abstratas, requerendo que esses métodos sejam definidos por qualquer filha.

Criando interfaces

Uma interface ajuda os desenvolvedores a forçar especificações de aplicação de forma rigorosa. É similar a uma classe abstrata, mas contém somente as assinaturas de método requeridas. Qualquer classe que implemente a interface deve também implementar todos os métodos de interface definidos.

Interfaces são definidas pelo uso da palavra-chave interface, e seus nomes são geralmente prefixados com a letra maiúscula `I`:

```
interface IMutant {
    public function getMutationLevel();
    public function setMutationLevel($mutationLevel);
}

class Xmen extends Hero implements IMutant {
// Esta classe deve implementar os métodos esboçados por IMutant
...
}
```

Namespaces

Namespaces no PHP são projetados para resolver o problema de escopo em bibliotecas PHP extensas. No Zend Framework 1, por exemplo, você encontra nomes extensos de classes, como `Zend\CodeGenerator\Php\Docblock\TagParam`. Esse tipo de nome era dado para evitar colisão com outro nome de classe.

Namespaces permitem que nomes iguais sejam definidos em contextos (namespaces) diferentes. Essa classe, por exemplo,

poderia se chamar apenas `Param`, se fosse definida com um namespace assim:

```
namespace Zend\CodeGenerator\Php\Docblock;
class Param {
}
```

Essa classe não entra em conflito com a seguinte:

```
namespace Acme\Physics\Rules;
class Param {
}
```

Embora tenham o mesmo nome, estão em contextos diferentes. Podemos, por exemplo, instanciar objetos de cada uma dessas classes assim:

```
$argument = new Zend\CodeGenerator\Php\Docblock\Param();
$criterion = new Acme\Physics\Rules\Param();
```

Entretanto, essa forma é claramente extensa. Por isso omitimos o namespace e usamos apelidos com o operador `use`, desta forma:

```
use Zend\CodeGenerator\Php\Docblock\Param;
use Acme\Physics\Rules\Param as Criterion;
$argument = new Param();
$criterion = new Criterion();
```

Quando o namespace não é definido para uma classe, por padrão, ela pertence ao namespace global. Vários namespaces podem ser importados ou apelidados com apenas um operador `use`, se forem separados por vírgula, conforme o exemplo a seguir.

```
use Zend\CodeGenerator\Php\Docblock\Param, Acme\Physics\Rules\Param as
Criterion;
$argument = new Param();
$criterion = new Criterion();
```

Late static bindings

Late Static Bindings resolve um problema que ocorre na herança de métodos estáticos. Quando uma classe-filha chama o método estático definido na classe-mãe, e esse método faz uma chamada a um método que é sobrescrito pela classe-filha, o código executado não é o da classe-filha, mas da classe-mãe.

Quer dizer, métodos chamados com a palavra-chave `self` são procurados na própria classe em que estão sendo invocados, mesmo que a invocação inicial tenha sido por uma classe herdeira. No exemplo a seguir, quando invocamos o método `whoAmI()` a partir da classe `Sheet`, o resultado é `Document`. Isso porque o método `getClassName()` invocado não é da classe `Sheet`, mas, sim, da `Document`, onde o método que a chama está definido.

```
class Document {
    public static function getClassName() {
        echo _ _CLASS_ _;
    }
    public static function whoAmI()     {
        self::getClassName();
    }
}
class Sheet extends Document {
    public static function getClassName() {
        echo _ _CLASS_ _;
    }
}
Sheet::whoAmI();
```

Para evitar esse resultado indesejado, substituímos a palavra `self` por `static`, como mostra o exemplo a seguir, que imprimirá `Sheet`.

```
class Document {
    public static function getClassName() {
        echo _ _CLASS_ _;
    }
    public static function whoAmI()     {
        static::getClassName();
    }
}
```

```
class Sheet extends Document {
    public static function getClassName() {
        echo _ _CLASS_ _;
    }
}
Sheet::whoAmI();
```

A constante `__CLASS__` retorna o nome da classe onde ela está contida. Se quisermos o nome da classe que executou a chamada ao método, temos de usar a função `get_called_class()` .

Traits

O PHP 5.4 disponibilizou a implementação de blocos de construção denominados traits. Um trait provê um conjunto de métodos que implementam comportamento. Classes e traits podem ser compostos de outros traits. Um trait tem uma estrutura similar a uma classe, mas não pode ser instanciado, apenas usado. E, ao contrário de uma classe, um trait pode ser usado por várias classes diferentes ao mesmo tempo. A sintaxe básica de um trait é como mostra o exemplo a seguir.

```
trait GenericTrait {
    public function doSomeSingle() {
        echo 'something single';
    }
}
```

Traits são usados por classes com a palavra-chave `use` , como no exemplo a seguir. Métodos definidos em traits sobrescrevem métodos de mesmo nome herdados pela classe. Métodos definidos na classe sobrescrevem métodos de mesmo nome definidos pelos traits usados pela classe. Isto é, os traits têm precedência sobre os métodos herdados, mas os métodos da classe têm precedência sobre os métodos dos traits.

```
class ClassC {
    use SpecialTrait;
}
```

Uma classe pode usar vários traits, basta separar os nomes com vírgulas no operador `use`. Eventualmente, traits diferentes podem ter métodos com nomes iguais. Você pode definir implicitamente qual dos métodos homônimos deseja usar, com a instrução `insteadof`.

Por exemplo, se os traits `Fifer`, `Fiddler` e `Practical` tiverem um método chamado `buildHouse()` e quisermos que a classe `PerfectPig` use o método do trait `Practical`, escreveremos assim:

```
class PerfectPig {
    use Fifer, Fiddler, Practical {
            Practical::buildHouse insteadof Fifer, Fiddler;
    }
}
```

E se, por acaso, quisermos usar dois métodos com o mesmo nome, podemos apelidar um com outro nome, conforme o exemplo a seguir:

```
class PerfectPig {
    use Fifer, Fiddler, Practical {
            Practical::buildHouse insteadof Fifer, Fiddler;
            Fifer::buildHouse as buildShitHouse;
    }
}
```

Um trait não é apenas uma interface com implementação, porque uma classe não pode implementar duas interfaces que tenham métodos com mesmo nome. Além disso, interfaces não suportam atributos, apenas constantes, e os métodos de uma interface têm de ser públicos.

Em traits, você pode definir atributos, inclusive estáticos, da mesma forma como faz em classes. Traits podem usar variáveis estáticas, só não podem defini-las. Classes também podem alterar a visibilidade de métodos de traits.

5.3 Data e hora

Função date()

A função `date()` é talvez uma das funções PHP mais frequentemente usadas; é capaz de recuperar cada atributo temporal de um *timestamp* específico. Todas as funções trabalham com data e hora em inglês, por padrão. Isso pode ser modificado com as funções que controlam a localização, mas, como isso é apenas uma revisão resumida, deixemos para outro capítulo.

Antes que você pergunte onde arrumar um timestamp, as funções que geram timestamps serão apresentadas adiante. No entanto, se você não sabe o que é um timestamp... Bem, o jeito é explicar.

Um timestamp é uma sequência de caracteres, denotando a data e/ou a hora na qual certo evento ocorreu. Esses dados são geralmente apresentados em um formato consistente, permitindo a fácil comparação de dois diferentes registros e o rastreamento do progresso ao longo do tempo; a prática de gravar timestamps de uma maneira consistente com os dados atuais é chamada *timestamping*.

Finalmente, o mais importante: para que serve um timestamp? Eles são tipicamente usados para registro (*logging*) de eventos, caso em que cada evento em um log é marcado com uma data e uma hora. Em sistemas de arquivo, timestamp pode significar a data/hora armazenada de criação ou modificação de um arquivo armazenado.

Agora já podemos usar a função `date()` com todos os seus parâmetros:

```
string date(string $format [, $int $timestamp])
```

Parâmetro	Descrição

Parâmetro	Descrição
a	Ante meridiem (am) e Post meridiem (pm) minúsculos.
A	Ante meridiem (AM) e Post meridiem (PM) maiúsculos.
B	Hora Swatch Internet.
c	Data ISO 8601.
e	Identificador de fuso horário.
g	Formato de hora 12-horas sem zeros à esquerda.
G	Formato de hora 24-horas com zeros à esquerda.
h	Formato de hora 12-horas com zeros à esquerda.
H	Formato de hora 24-horas com zeros à esquerda.
i	Minutos com zeros à esquerda.
I (i maiúsculo)	Especifica se a data está no horário de verão.
O	Diferença para a hora de Greenwich (GMT) em horas.
P	Diferença para a hora de Greenwich (GMT) com dois pontos (:) entre as horas e os minutos.
r	Data RFC 2822.
s	Segundos, com zeros à esquerda.
T	Abreviação do fuso horário.
u	Milissegundos.
U	Segundos desde a Era Unix.
z	Intervalo do fuso horário em segundos.

Parâmetros de dia

Parâmetro	Descrição
d	Dia do mês, dois dígitos com zeros à esquerda.
D	Três letras representando o dia.
j	Dias do mês sem zeros à esquerda.
l (de laranja)	Representação textual do dia.
N	Representação numérica ISO-8601.
S	Dois caracteres com o sufixo original em inglês para o dia do mês.
w	Representação numérica do dia da semana.
z	Intervalo numérico do dia do ano.

Parâmetros de semana

Parâmetro	Descrição
W ISO-8601	Número da semana no ano.
F	Representação por extenso do mês.
m	Representação numérica do mês.

Parâmetros de mês

Parâmetro	Descrição
M	Representação do mês com três letras.
n	Representação numérica do mês sem zeros à esquerda.
t	Número de dias de um dado mês.

Parâmetros de ano

Parâmetro	Descrição
L	Se o ano é bissexto.
o	Número do ano ISO-8601.
Y	Representação numérica completa do ano.
y	Representação do ano com dois dígitos.

Exemplos de funções de data

Para tornar os exemplos mais sintéticos, usaremos algumas frases em inglês, pois as funções retornam nomes de dias da semana e meses do ano na língua de William Shakespeare por padrão.

- Exibir “September 7, 2008”:

```
print date('F j, Y');
```

- Exibir “9/7/08”:

```
print date('m/j/y');
```

- Exibir “Today is Thursday, October 23 20:31:43pm”:

```
printf("Today is %s",date('l, F j h:i:sa'));
```

- Exibir “There are 31 days in October”:

```
printf("There are %d days in %s.", date('t'),date('F'));
```

Configurando o fuso horário

Você pode configurar o fuso horário para todos os scripts pela configuração da diretiva `date.timezone`, dentro do arquivo `php.ini`, ou em um script base usando a função `date_default_timezone_set()`.

Outras funções úteis

Função	O que faz
`mktime`	Retorna o timestamp Unix para uma determinada data.
Assinatura	`int mktime ([ int $hora [, int $minuto [, int $second [, int $mes [, int $dia [, int $ano [, int $is_dst ]]]]]]] )`
time	Retorna o timestamp atual.
Assinatura	`int time ( void )`
`setlocale`	Configura a localização do script.
Assinatura	`string setlocale ( mixed $category, string $locale [, string $... ] )`
`strtotime`	Converte uma data/hora textual em inglês para um timestamp Unix.
Assinatura	`int strtotime ( string $time [, int $now ] )`
`checkdate`	Valida a data composta dos argumentos `$month` , `$day` e `$year` .
Assinatura	`bool checkdate ( int $mês, int $dia, int $ano )`
`getdate`	Recupera um timestamp como um array associativo. Chaves associativas incluem `seconds` (segundos), `minutes` (minutos), `hours` (horas), `mday` (dia do mês), `wday` (dia da semana), `month` (mês), `year` (ano), `yday` (dia do ano), `weekday` (dia da semana por extenso), `month` (mês por extenso) e `0` (segundos desde a Era Unix).
Assinatura	`array getdate ([ int $timestamp ] )`

O PHP 5.1.0 introduziu uma classe `DateTime` (logicamente orientada a objetos).

Exemplos relacionados à data

- Exibir “October 23 falls on a Thursday”:

```
$date = date('l',mktime(0,0,0,10,23,2008));
printf("October 23 falls on a %s", $date);
```

- Exibir “Next month is November”:

```
printf("Next month is %s", date('F', strtotime('+1 month')));
```

- Exibir “Last Friday fell on October 17, 2008”:

```
$date = date('F d, Y', strtotime('Last Friday'));
printf("Last Friday fell on %s", $date);
```

- Exibir “Hoje é quinta-feira”:

```
setlocale(LC_ALL, "ptb");
printf("Hoje &eacute; %s", strftime("%A"));
```

- Recuperar a data da última modificação de uma página:

```
echo date('l, F j h:i:sa', filemtime($_SERVER["SCRIPT_NAME"]));
```

- Calcular a diferença entre duas datas:

```
$date1 = strtotime("2008-08-14");
$date2 = strtotime("2008-07-11");
$diff = $date2 - $date1;
printf("Difference in days: %s", $diff / 60 / 60 / 24);
```

Formatação de datas

A classe `DateTime` pode ser usada para representar uma data e uma hora específica e para executar operações sobre valores de data e hora. Para instanciar um novo objeto da classe `DateTime` , forneça a data/hora em um formato válido. Você pode opcionalmente fornecer o fuso horário como segundo parâmetro.

O método `DateTime::format()` permite que uma data/hora seja formatada de um modo mais elegante do que por meio de caracteres de formatação, como vimos anteriormente. A classe `DateTime` inclui várias constantes predefinidas para formatos comuns. Para imprimir, por exemplo, uma data/hora no formato ISO 8601:

```
$date = new DateTime('2013-1-15');
echo $date->format(DateTime::ISO8601);
// imprimirá 2013-01-15T00:00:00+0100
```

Aritmética de datas

PHP também inclui uma classe `DateInterval` que pode ser usada para adicionar ou subtrair datas, assim como para determinar a diferença entre duas datas. Essa aritmética pode ser feita com `DateTime::add()`, `DateTime::sub()` e `DateTime::diff()`, respectivamente. A seguir, temos um exemplo de como subtrair 30 dias de uma data:

```
$date = new DateTime('2013-2-28');
$interval = new DateInterval('P30D');
$date->sub($interval);
print_r($date->format(DateTime::ISO8601)); // 2013-01-29T00:00:00+0100
```

A string `P30D` significa 30 dias. Para determinar um intervalo de três meses, por exemplo, escrevemos `P3M`, e um intervalo de três anos é descrito com `P3Y`.

A seguir, temos um exemplo de como obter a diferença em dias entre duas datas:

```
$date1 = new DateTime('2013-2-28');
$date2 = new DateTime('2013-01-29');
echo $date2->diff($date1)->format('%d'); // imprime 30
```

5.4 Diretórios e arquivos

Funções para diretórios

Função	Descrição
`chdir`	Muda o diretório.
`chroot`	Muda o diretório raiz (root).
`closedir`	Fecha o recurso manipulador de diretório.
`dir`	Retorna uma instância da classe `Directory`.
`getcwd`	Retorna o diretório atual.
`opendir`	Abre um recurso manipulador de diretório.
`readdir`	Retorna os campos lidos de um recurso manipulador de diretório para a posição atual.
`rewinddir`	Posiciona o ponteiro de leitura do recurso manipulador de diretório para o início.
`scandir`	Retorna a lista de arquivos e diretórios no caminho especificado.

Esquema geral para abrir e ler diretórios entrada a entrada

```
$handle = opendir($path);
while($entry = readdir($handle)) {
    // faz alguma coisa com $entry, que é um texto
}
closedir($handle);
```

Funções para sistema de arquivos

Função	Descrição
`basename`	Retorna parte do nome do arquivo de um caminho de diretório.

Função	Descrição
`chgrp`	Modifica o grupo ao qual o arquivo pertence.
`chmod`	Modifica as permissões do arquivo.
`chown`	Modifica o proprietário do arquivo.
`clearstatacache`	Limpa as informações em cache sobre arquivos.
`copy`	Copia um arquivo.
`dirname`	Retorna apenas o diretório de um caminho de diretório, sem o nome do arquivo.
`disk_free_space`	Retorna o espaço disponível no diretório especificado em bytes.
`disk_total_space`	Retorna o tamanho total do diretório especificado em bytes.
`fclose`	Fecha um recurso de arquivo aberto.
`feof`	Retorna `true` se o final do arquivo aberto for alcançado.
`fflush`	Força a liberação do buffer para um arquivo.
`fgetc`	Retorna um caractere lido da posição atual do ponteiro do arquivo.
`fgetcsv`	Retorna uma linha lida da posição atual do ponteiro do arquivo no formato CSV.
`fgets`	Retorna uma linha lida da posição atual do ponteiro do arquivo como texto.
`fgetss`	Retorna uma linha lida da posição atual do ponteiro do arquivo sem tags HTML.

Função	Descrição
`file_exists`	Retorna `true` se o arquivo especificado existe.
`file_get_contents`	Retorna um texto com todo o conteúdo do arquivo especificado.
`file_put_contents`	Grava um texto no arquivo especificado.
`file`	Retorna todo o conteúdo de um arquivo como array.
`fileatime`	Retorna o último horário em que o arquivo foi acessado como Unix timestamp.
`filectime`	Retorna o horário em que o inode do arquivo foi modificado como Unix timestamp.
`filegroup`	Retorna o id do grupo ao qual o arquivo pertence. Use `posix_getgtgid()` para resolvê-lo.
`fileinode`	Retorna o valor do inode do arquivo.
`filemtime`	Retorna o horário em que o arquivo foi modificado como Unix timestamp.
`fileowner`	Retorna o id do proprietário do arquivo. Use `posix_getpwuid()` para resolvê-lo.
`fileperms`	Retorna um inteiro representando as permissões do arquivo (como 1777 ou 0644).
`filesize`	Retorna o tamanho do arquivo especificado em bytes.
`filetype`	Retorna o tipo do arquivo especificado.

Função	Descrição
`flock`	Tenta travar um arquivo e retorna `true` se conseguir. A trava é liberada por `fclose()`.
`fnmatch`	Compara um nome de arquivo com um padrão e retorna `true` se casar.
`fopen`	Abre um arquivo e retorna um recurso para manipulação.
`fputcsv`	Grava um array como texto CSV na posição atual do ponteiro do arquivo.
`fread`	Retorna uma quantidade de bytes lidos da posição atual do ponteiro do arquivo.
`fscanf`	Retorna uma linha de um arquivo interpretada de acordo com o padrão especificado.
`fseek`	Move o ponteiro do arquivo para a posição especificada.
`fstat`	Retorna um array com estatísticas do arquivo aberto.
`ftell`	Retorna a posição do ponteiro de arquivo.
`ftruncate`	Reduz um arquivo para o tamanho especificado (não compacta conteúdo, apaga).
`fwrite`	Grava o texto na posição atual do arquivo especificado.
`glob`	Retorna um array com caminhos de arquivo e diretório que casam com um padrão.

Função	Descrição
`is_dir`	Retorna `true` se o caminho especificado for um diretório.
`is_executable`	Retorna `true` se o arquivo especificado tiver permissão de execução.
`is_file`	Retorna `true` se o caminho especificado for um arquivo.
`is_link`	Retorna `true` se o caminho especificado for um link simbólico.
`is_readable`	Retorna `true` se o arquivo especificado tiver permissão de leitura.
`is_uploaded_file`	Retorna `true` se o arquivo especificado for enviado por HTTP `POST`.
`is_writable`	Retorna `true` se o arquivo tiver permissão de escrita.
`lchgrp`	Altera o grupo proprietário de um link simbólico.
`lchown`	Altera o proprietário de um link simbólico.
`link`	Cria um hard link.
`lstat`	Retorna um array com informações sobre um link simbólico.
`mkdir`	Cria um diretório.
`move_uploaded_file`	Move um arquivo enviado por HTTP `POST` para um novo local.
`parse_ini_file`	Retorna o conteúdo de arquivo de configuração INI como um array.

Função	Descrição
`parse_ini_string`	Retorna um texto no padrão INI como um array.
`pathinfo`	Retorna um array associativo contendo informações sobre o caminho especificado.
`pclose`	Fecha o recurso para um processo aberto por `popen()`.
`popen`	Abre um recurso um processo.
`readfile`	Lê um arquivo e escreve seu conteúdo para o buffer de saída.
`readlink`	Retorna o alvo de um link simbólico.
`realpath_cache_get`	Retorna um array de entradas de cache, geradas por `realpath()`.
`realpath_cache_size`	Retorna a quantidade de memória usada pelo cache, gerada por `realpath()`.
`realpath`	Retorna o caminho absoluto a partir do caminho relativo.
`rename`	Renomeia um arquivo ou diretório.
`rewind`	Altera a posição do ponteiro de arquivos para o início.
`rmdir`	Remove um diretório.
`stat`	Retorna um array com estatísticas sobre um arquivo.
`symlink`	Cria um link simbólico.
`tempnam`	Cria um arquivo com nome único e retorna seu nome.

Função	Descrição
`tmpfile`	Cria um arquivo temporário e retorna o recurso manipulador.
`unlink`	Apaga o arquivo especificado.

Esquema geral para abrir e ler arquivos linha a linha

```
$handle = fopen($path, $mode);
while (!feof($handle)) {
    $row = fread($handle, $length);
    // faz alguma coisa com $row...
}
fclose($handle);
```

5.5 Monitoração e medição em segundo plano

Existe uma estrutura de controle chamada declare no PHP que permite que funções sejam executadas em segundo plano durante a execução de um determinado bloco de código. É uma construção útil para fazer monitoração de estados em aplicações e medição de recursos de sistema para aferição do desempenho da aplicação.

O script a ser monitorado/medido deve estar dentro de um bloco, assim:

```
declare(ticks=1) {
    // seu código
}
```

A cada `tick` (um evento que ocorre para cada instrução de baixo nível executada pelo analisador sintático), são executadas as funções registradas com `register_tick_function()`. Para remover uma função da pilha de execução do bloco, declare `use unregister_tick_function()`.

5.6 Servidor embutido

PHP 5.4 introduziu um servidor web embutido para ser usado em desenvolvimento (e somente para esse propósito). Para iniciar o servidor PHP, abra o terminal, navegue até o diretório raiz do seu projeto e digite o seguinte comando:

```
php -S localhost:8080
```

5.7 Modo interativo

Na linha de comando, PHP pode ser executado no modo interativo, que permite a interpretação imediata de instruções ou estruturas de controle. É uma ferramenta extremamente útil para elucidar dúvidas sobre o resultado de funções sozinhas ou combinadas. O modo interativo é iniciado com o parâmetro `-a` .

```
php -a
```

5.8 Standard PHP Library (SPL)

A SPL provê funções e classes para resolver problemas extremamente genéricos, que ocorrem na maioria dos softwares, com um foco maior em software orientado a objetos. As classes oferecidas pela SPL têm a vantagem de não precisarem ser carregadas, pois já fazem parte de uma extensão PHP, codificadas em linguagem C. Isso implica que as classes da SPL apresentam um desempenho superior a classes implementadas em PHP.

SPL Datastructures

Uma estrutura de dados é um tipo de dado de alto nível implementado com tipos de dados existentes e usado como

ferramenta para solucionar problemas recorrentes em programas de computadores (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

SplDoublyLinkedList

A classe `SplDoublyLinkedList` é uma implementação da estrutura de dados lista linear duplamente ligada. Segundo Tenenbaum, Langsam e Augenstein (1995, p. 224), uma lista ligada linear é uma estrutura na qual “cada item na lista é chamado nó e contém dois campos, um campo de informação e um campo do endereço seguinte. O campo de informação armazena o real elemento da lista”. Na lista linear duplamente ligada, o nó tem um campo a mais, que é o nó do endereço anterior.

SplStack

A classe `SplStack` implementa uma pilha. De acordo com Tenenbaum, Langsam e Augenstein (1995, p. 86), “uma pilha é um conjunto ordenado de itens no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados itens em uma extremidade chamada topo da pilha”.

SplQueue

A classe `SplQueue` é uma implementação de uma fila. Tenenbaum, Langsam e Augenstein (1995, p. 207) afirmam que “uma fila é um conjunto ordenado de itens a partir do qual podem-se eliminar itens numa extremidade (chamada início da fila) e no qual podem-se inserir itens na outra extremidade (chamada final da fila)”.

SplHeap

A classe `SplHeap` implementa uma árvore binária. De acordo com Tenenbaum, Langsam e Augenstein (1995, p. 303) “uma árvore binária é um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos disjuntos. O primeiro subconjunto contém um único elemento, chamado raiz da árvore. Os outros dois subconjuntos são árvores binárias, chamadas subárvores esquerda

e direita da árvore original. Uma subárvore esquerda ou direita pode estar vazia. Cada elemento de uma árvore binária é chamada nó da árvore.”

SplFixedArray

O array PHP é elástico. Podemos adicionar quantos elementos quisermos. Ele também pode ser usado como um mapa de dados, aceitando strings como chaves. `SplFixedArray` cria um array de tamanho fixo, que admite apenas índices numéricos.

SplObjectStorage

Essa classe gera um objeto que agrega vários objetos.

SPL Iterators

A estrutura `foreach` pode percorrer os elementos de um objeto, desde que sua classe implemente a interface `Iterator` . A SPL possui várias implementações da interface `Iterator` , que define os métodos que uma classe precisa ter para ser iterável (para que seja possível percorrer elementos agregados a ela de forma sequencial).

Classe	Descrição
`AppendIterator`	Itera sobre vários iteradores um após o outro.
`ArrayIterator`	Permite destruir e modificar valores e chaves enquanto itera.
`CachingIterator`	Suporta iteração com cache.
`CallbackFilterIterator`	Aceita um callback em seu construtor.
`DirectoryIterator`	Itera sobre diretórios de sistemas de arquivos.

Classe	Descrição
`EmptyIterator`	Um iterador vazio.
`FilesystemIterator`	Itera sobre o sistema de arquivos.
`FilterIterator`	Filtra valores indesejados.
`GlobIterator`	Produz um efeito similar à função `glob()`.
`InfiniteIterator`	Permite iterar infinitamente sem ter de voltar ao início da coleção.
`IteratorIterator`	Permite converter um `Traversable` em `Iterator`.
`LimitIterator`	Permite iterar sobre um subconjunto limitado de itens.
`MultipleIterator`	Itera sequencialmente sobre todos os iteradores anexados.
`NoRewindIterator`	Não permite retorno ao início (primeiro item) da coleção.
`ParentIterator`	Extensão de `FilterIterator` que usa `RecursiveIteratorIterator`.
`RecursiveArrayIterator`	Faz o mesmo que `ArrayIterator` de forma recursiva.
`RecursiveCachingIterator`	Faz o mesmo que `CachingIterator` de forma recursiva.
`RecursiveCallbackFilterIterator`	Faz o mesmo que `CallbackFilterIterator` de forma recursiva.

Classe	Descrição
`RecursiveDirectoryIterator`	Faz o mesmo que `DirectoryIterator` de forma recursiva.
`RecursiveFilterIterator`	Faz o mesmo que `FilterIterator` de forma recursiva.
`RecursiveIteratorIterator`	Faz o mesmo que `IteratorIterator` de forma recursiva.
`RecursiveRegexIterator`	Pode filtrar outro iterador recursivo via expressão regular.
`RecursiveTreeIterator`	Itera sobre um `RecursiveIterator` gerando uma árvore ASCII gráfica.
`RegexIterator`	Pode filtrar outro iterador via expressão regular.

SPL Interfaces

Gamma et al. (2000, p. 33) definem um princípio para um projeto de software reutilizável e orientado a objetos: “Programe para uma interface, não para uma implementação. Não declare variáveis como instâncias de classes concretas específicas. Em vez disso, prenda-se somente a uma interface definida por uma classe abstrata”.

Em termos práticos, em vez de:

```
/**
* @var ClassName
*/
private $attribute;
```

Faça:

```php
/**
* @var InterfaceName
 */
private $attribute;
```

Interface	Descrição
`Countable`	Classes com essa interface podem ser usadas com a função `count()`.
`OuterIterator`	Define métodos para iterar sobre iteradores.
`RecursiveIterator`	Define métodos para iterar sobre iteradores recursivamente.
`SeekableIterator`	Extensão de `Iterator` que adiciona o método `seek()`.

SPL Exceptions

A criação de exceções específicas permite saber exatamente que tipo de problema ocorreu e assim saber se é possível tratá-lo, ou se ele deve ser encaminhado para o próximo método na pilha de execução de chamadas.

Classe	Exceção que trata
`BadFunctionCallException`	Falha em chamada à função.
`BadMethodCallException`	Falha em chamada ao método.
`DomainException`	Falha de domínio.
`InvalidArgumentException`	Argumento inválido.
`LengthException`	Falha de tamanho.
`LogicException`	Falha de lógica.
`OutOfBoundsException`	Falha de ligações ultrapassadas.

Classe	Exceção que trata
`OutOfRangeException`	Falha de limites ultrapassados.
`OverflowException`	Falha de valor acima do limite superior.
`RangeException`	Falha de intervalo.
`RuntimeException`	Falha de tempo de execução.
`UnderflowException`	Falha de valor abaixo do valor inferior.
`UnexpectedValueException`	Falha de valor inesperado.

SPL File Handling

Classe	Descrição
`SplFileInfo`	Provê informações sobre um arquivo individual.
`SplFileObject`	Uma interface orientada a objetos para um arquivo.
`SplTempFileObject`	Uma interface orientada a objetos para um arquivo temporário.

SPL Miscellaneous

Classe/Interface	Descrição
`ArrayObject`	Encapsula um array.
`SplObserver`	Define a interface para uma classe observadora/ouvinte.
`SplSubject`	Define a interface para uma classe observada/ouvida.

Aqui encerramos a apresentação dos instrumentos básicos de reutilização de código, disponíveis em PHP. O desenvolvedor PHP pode reutilizar código na forma de funções, de classes ou de ambos, conforme seja mais conveniente.

Antes de criar uma função, verifique se o PHP já não possui uma que resolva o problema. Elas estão presentes desde as primeiras versões da linguagem e, atualmente, há diversas extensões com funções que trabalham com os mais variados campos de conhecimento.

A SPL provê ao programador as abstrações e interfaces mais genéricas para aplicações orientadas a objetos. Mas são os componentes e frameworks implementados a partir de padrões de projetos que proporcionam o mais alto nível de reúso. E é por isso que, nos próximos capítulos, vamos conhecer componentes e seu agrupamento na forma de um framework específico.

Capítulo 6
MVC e MVVM com Laminas

Aquelas tarefas de programação que duram a noite toda fazem você se sentir o maior programador do mundo; depois é preciso passar várias semanas corrigindo os defeitos instalados durante o seu momento de glória. – Steve McConnel

MVC é o acrônimo para Model-View-Controller, o padrão de arquitetura que vimos no capítulo 1. MVVM é um outro padrão, relacionado e (talvez, de certa forma) derivado do MVC, sobre o qual falaremos no momento apropriado. O importante agora é saber que ambos serão utilizados no exercício que iniciaremos aqui.

Neste capítulo, aplicamos o padrão MVC com o componente Laminas\Mvc e o padrão MVVM com o componente Laminas\View\Model\ViewModel, ambos do Laminas. Aprenderemos a dar o domínio de nossa aplicação ao framework, confiando a ele o controle sobre o nosso código. Conheceremos ainda o componente de geração de formulários dinâmicos, o Laminas\Form.

Aqui não vamos apenas tirar o coelho da cartola, mas desvendaremos a mágica por trás das cortinas explicando o que está acontecendo e qual é a ideia por trás da implementação Laminas. Cuidado! Você pode aprender algo neste capítulo!

6.1 Criação do projeto

Sua ansiedade por implementar pode tê-lo levado a pular os capítulos iniciais e vir direto a este. Se as questões de boas práticas de desenvolvimento não são problemas para você, e seu domínio de programação orientada a objetos e padrões de projeto é maduro para não se perder nas implementações que usam esses conceitos,

então podemos dar início a um projeto PHP com a estrutura Laminas.

Os capítulos iniciais estarão sempre esperando pela sua consulta, caso alguma dúvida conceitual surja. Um alerta, entretanto, é necessário: se alguma referência ao ambiente de desenvolvimento não estiver clara, não haverá outra saída a não ser ler o capítulo sobre isto.

Uma funcionalidade básica de sistemas de informação que permanece independente das tecnologias utilizadas é o cadastro. O cadastro é uma macrofuncionalidade, composta por quatro operações: criação, recuperação, atualização e destruição.

O que será criado, o que será recuperado, atualizado ou destruído não importa para nós. Isso é regra de negócio do cliente. O que nos importa é como faremos isso acontecer. Podemos cadastrar produtos, podemos cadastrar animais de estimação, podemos cadastrar armas ou fragmentos de rochas. Não nos importa o conteúdo a ser armazenado, pois isso é o insumo que o cliente vai nos fornecer. Então, vamos praticar o exercício da generalidade. Se a professora usar uma letra diferente no lugar do x, devemos ser capazes de resolver a equação pelo método que ela nos ensinou.

Imagine que você recebe uma demanda de uma organização espacial denominada de *Guardiões do Universo*. Essa organização administra um serviço policial intergaláctico chamado *Tropa dos Lanternas Verdes*. Lanterna Verde é o nome dado ao agente dessa força, por ele portar um anel e um uniforme com a imagem de uma lanterna da cor verde. Os Lanternas Verdes patrulham áreas denominadas de Setores Espaciais. Cada setor possui um grupo de Lanternas para patrulhá-lo. Os Guardiões do Universo, que são seus clientes, precisam de um sistema de controle e monitoração da Tropa dos Lanternas Verdes. A primeira entrega é o cadastro dos Setores Espaciais e dos Lanternas.

Por que não fazemos um cadastro de discos e categorias de músicas? Porque não é o que nosso cliente pediu. O fato de desconhecer o negócio nos impedirá de criar a funcionalidade? Não, porque a funcionalidade pedida é genérica. Haverá um momento em que teremos de lidar com funcionalidades que não são genéricas, mas só então precisaremos fazer perguntas aos nossos clientes.

Criando um novo projeto com Composer

Primeiro vamos criar o esqueleto do projeto usando o Composer. O Composer, que já instalamos, é um gerenciador de dependências para projetos PHP. Ele localiza, instala e atualiza bibliotecas de classes PHP.

Baixe-o a partir de https://getcomposer.org/composer.phar, renomeie o arquivo para `composer` somente, e coloque-o em um diretório que esteja no caminho de busca do sistema operacional, para que ele seja encontrado a partir de qualquer diretório. Por que renomear? Para não ter de digitar sempre a extensão do arquivo e conferir-lhe a aparência de um comando do nosso ambiente de desenvolvimento.

Com o Composer instalado, vamos usar o comando de criação de projeto a partir de um modelo. No terminal, no diretório `htdocs` , digite o seguinte comando:

```
[DIRETÓRIO ONDE O COMPOSER FOI INSTALADO]/composer create-project -sdev
laminas/laminas-mvc-skeleton corps1
```

Esse comando baixará os componentes básicos do Laminas para criar para uma aplicação PHP, conforme o trecho de saída a seguir, além de criar a estrutura de pastas e arquivos.

```
Creating a "laminas/laminas-mvc-skeleton" project at "./corps1"
Installing laminas/laminas-mvc-skeleton (dev-master
df7ec944ce1a041b155ed64ddbf31431225e2a04)
  - Installing laminas/laminas-mvc-skeleton (dev-master df7ec94): Cloning
df7ec944ce from cache
Created project in /tmp/corps1
```

```
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 20 installs, 0 updates, 0 removals
  - Installing laminas/laminas-zendframework-bridge (1.0.4): Downloading
(connecting.Downloading (100%)
  - Installing laminas/laminas-component-installer (2.2.0): Downloading
(connecting..Downloading (100%)
  - Installing laminas/laminas-skeleton-installer (0.1.7): Downloading
(connecting...Downloading (100%)
  - Installing laminas/laminas-development-mode (3.2.0): Downloading
(100%)
  - Installing laminas/laminas-stdlib (3.2.1): Downloading (100%)
  - Installing laminas/laminas-loader (2.6.1): Downloading (100%)
  - Installing laminas/laminas-eventmanager (3.2.1): Downloading (100%)
  - Installing laminas/laminas-json (3.1.2): Downloading (100%)
  - Installing laminas/laminas-view (2.11.4): Downloading (100%)
  - Installing psr/container (1.0.0): Downloading (100%)
  - Installing container-interop/container-interop (1.2.0): Downloading
(connecting..Downloading (100%)
  - Installing laminas/laminas-servicemanager (3.4.1): Downloading (100%)
  - Installing laminas/laminas-validator (2.13.4): Downloading (100%)
  - Installing laminas/laminas-escaper (2.6.1): Downloading (100%)
  - Installing laminas/laminas-uri (2.7.1): Downloading (100%)
  - Installing laminas/laminas-http (2.12.0): Downloading (100%)
  - Installing laminas/laminas-router (3.3.2): Downloading (100%)
  - Installing laminas/laminas-config (3.3.0): Downloading (100%)
  - Installing laminas/laminas-modulemanager (2.8.4): Downloading (100%)
  - Installing laminas/laminas-mvc (3.1.1): Downloading (100%)
```

Ao final do processo de criação do projeto, as últimas linhas devem ser estas:

```
Writing lock file
Generating autoload files
4 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

Após a execução do Composer, a pasta `corps1` estará criada com o seguinte conteúdo:

- `config` : configuração da aplicação;

- `data` : dados da aplicação;
- `module` : módulos da aplicação;
- `public` : diretório público, com estilos, imagens e o arquivo `index.php` ;
- `vendor` : bibliotecas de classes – é onde ficam os componentes do Laminas.

A primeira alteração que você fará, antes de prosseguirmos, será desabilitar o cache do projeto, para garantirmos que alterações nos arquivos de configuração terão efeito de forma imediata. Edite o arquivo `application.config.php` dentro da pasta `config` , e altere o valor das chaves `config_cache_enabled` e `module_map_cache_enabled` para `false` .

Configurando projeto no Eclipse

A estrutura do projeto foi criada pelo Composer. Agora vamos abrir essa estrutura como um projeto do Eclipse. Como não criamos o projeto pelo Eclipse, ele não o reconhecerá inicialmente como tal – para ele, é apenas um diretório qualquer no sistema de arquivos.

Por isso, criaremos um projeto genérico do Eclipse associado ao diretório criado pelo Composer e, em seguida, especificaremos que se trata de um projeto PHP e, mais especificamente, de um projeto PHP construído com Laminas.

Para isso, acesse o menu `File ? New ? PHP Project` , de acordo com a figura a seguir.

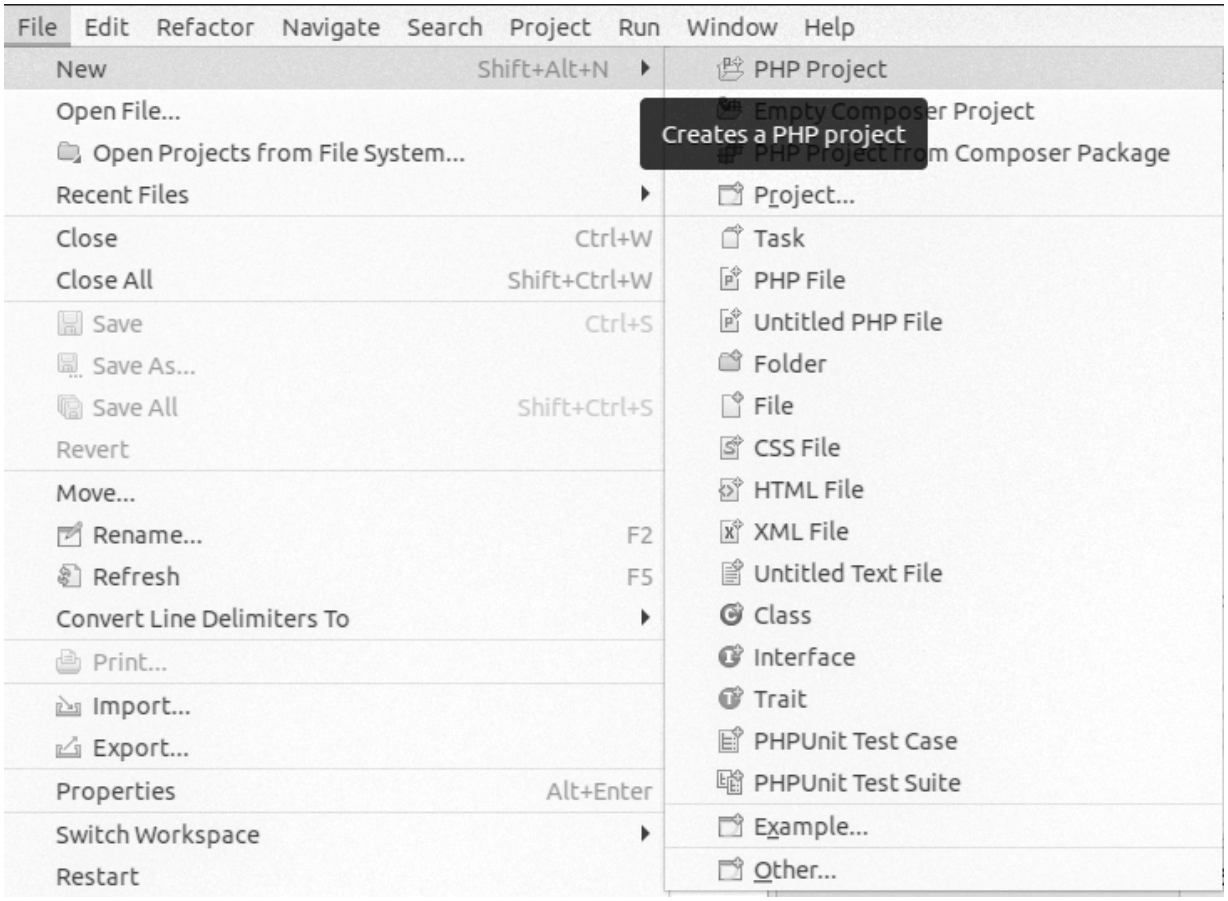


Figura 6.1: Criação de novo projeto no Eclipse PDT

Será aberta a janela de criação de projeto, onde colocaremos o nome do projeto (`corps1`).

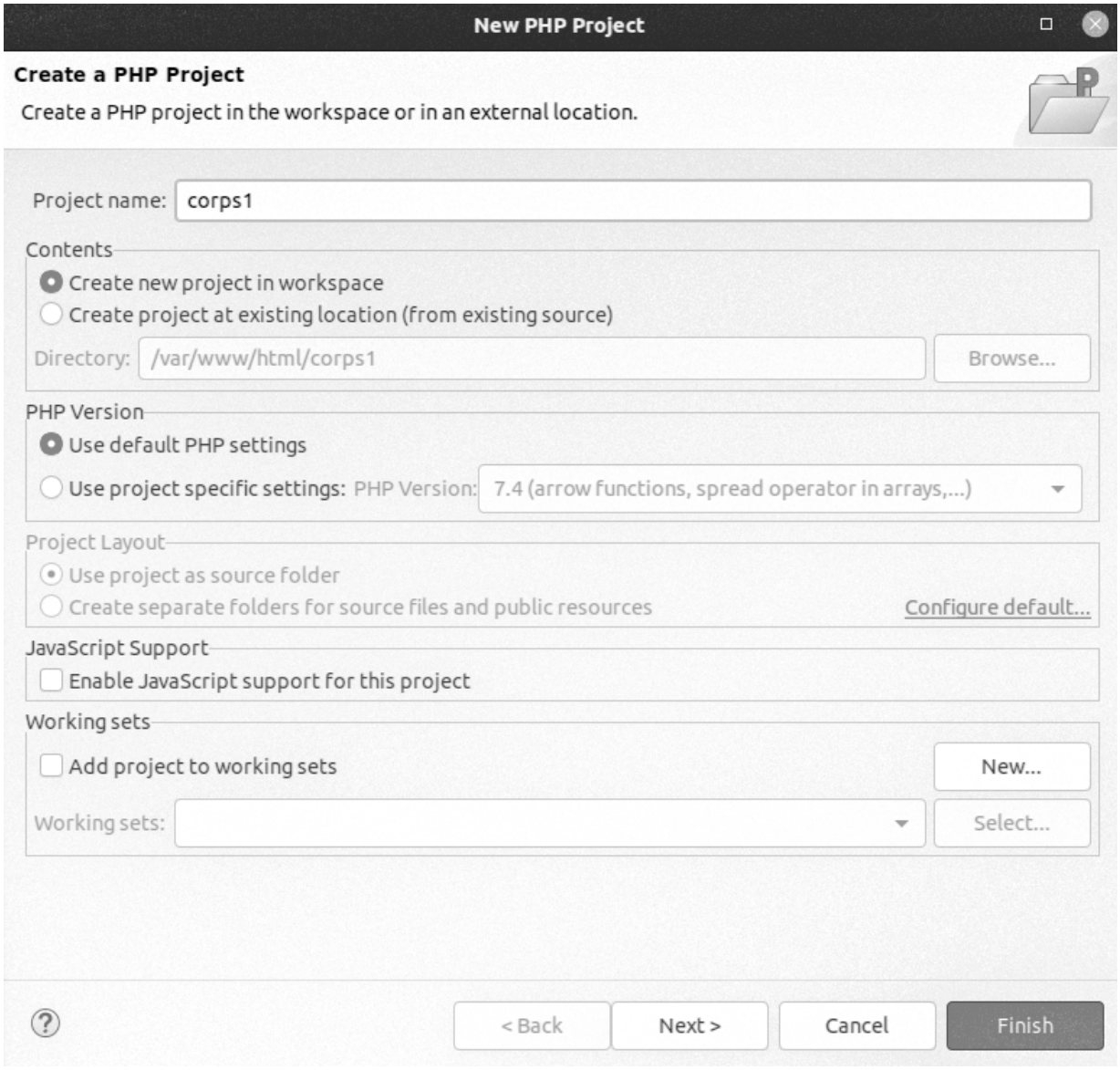


Figura 6.2: Janela New Project

Pode clicar no botão `Finish` e o projeto será criado (ou melhor, a pasta existente será identificada pelo Eclipse como um projeto). Na visão da esquerda, chamada `Project Explorer`, será exibido um nó com o nome `corps1`.

O Eclipse PDT tem um executável do Composer embutido. O Composer, que já instalamos, é um gerenciador de dependências para projetos PHP. Ele localiza, instala e atualiza bibliotecas de classes PHP, controlando as versões a partir de um arquivo

`composer.json`. Observe que o projeto `corps1` tem um arquivo `composer.json` em seu diretório raiz. Clique duas vezes sobre o arquivo para abrir a janela de edição do Composer.

Para atualizar as dependências do projeto, você pode utilizar o item `Update Dependencies` da visão `General`, conforme a próxima figura. Ele abstrai a execução do Composer. Porém, nada impede que você execute o Composer diretamente. Afinal, é sempre bom ter um plano B. Outro motivo é que você pode precisar executar um Composer mais recente do que o embutido no Eclipse, como é o caso do que baixamos anteriormente.

Se você digitar a URL http://localhost/corps1/public no navegador, deverá ter como resultado a página mostrada na figura a seguir.

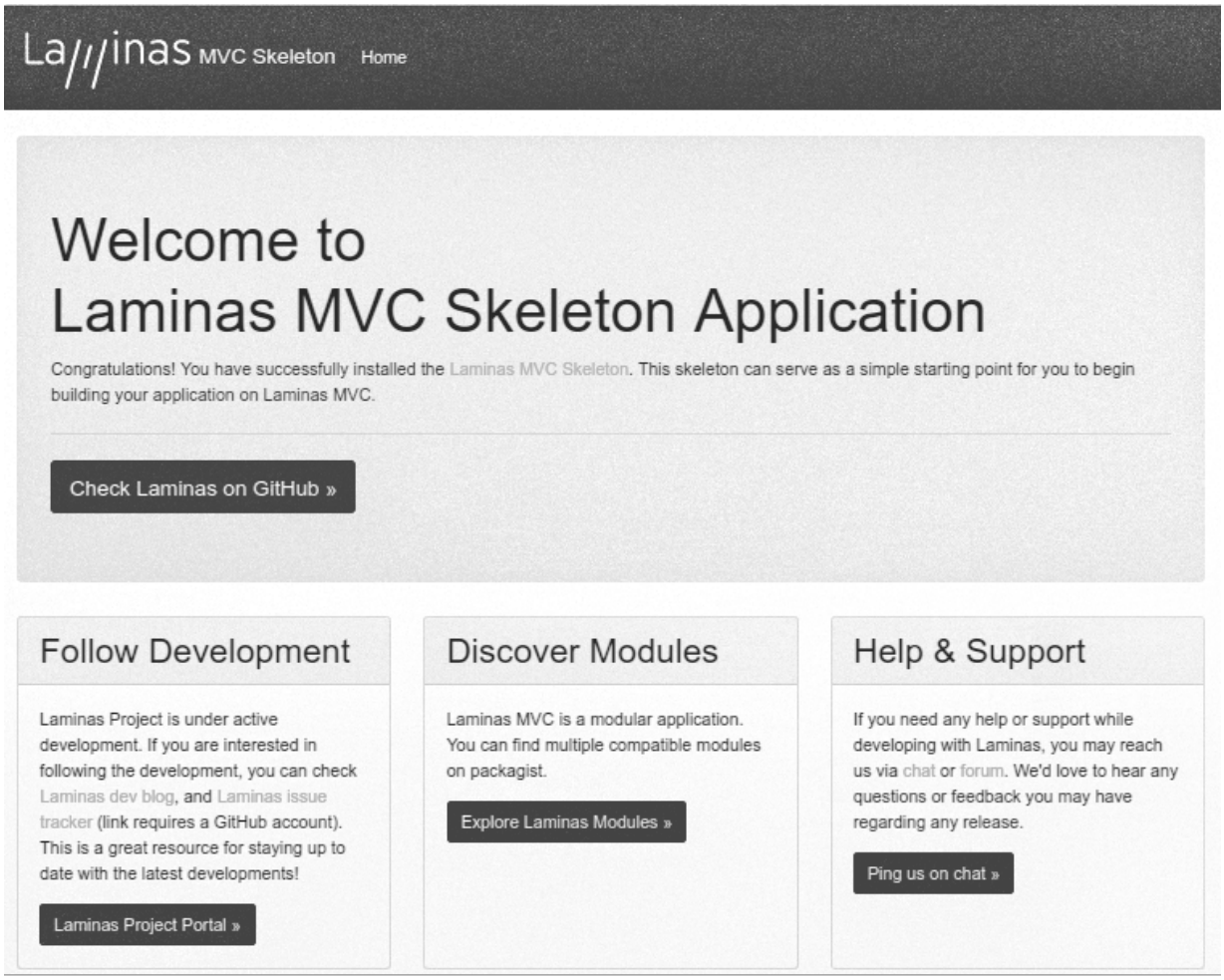

Laminas MVC Skeleton
Home
Welcome to
Laminas MVC Skeleton Application
Congratulations! You have successfully installed the Laminas MVC Skeleton. This skeleton can serve as a simple starting point for you to begin building your application on Laminas MVC.
Check Laminas on GitHub »
Follow Development
Laminas Project is under active development. If you are interested in following the development, you can check Laminas dev blog, and Laminas issue tracker (link requires a GitHub account). This is a great resource for staying up to date with the latest developments!
Laminas Project Portal »
Discover Modules
Laminas MVC is a modular application. You can find multiple compatible modules on packagist.
Explore Laminas Modules »
Help & Support
If you need any help or support while developing with Laminas, you may reach us via chat or forum. We'd love to hear any questions or feedback you may have regarding any release.
Ping us on chat »

Figura 6.3: Homepage padrão de projeto Laminas

Mas esta não será a URL que vamos usar para acessar o sistema. Ele tem de ser acessado por uma URL que use a pasta `public` como `webroot`. A `webroot` é a pasta considerada como o ponto inicial das páginas ou aplicações web em um servidor web. Então, vamos utilizar o servidor embutido do PHP para levantar a aplicação e definir um webroot diferente.

Execute este comando, a partir do diretório `corps1`:

```
php -S localhost:8000 -t public
```

Assim, você pode acessar a aplicação pela URL http://localhost:8000. Doravante, quando nos referirmos ao nome das aplicações (`corps1` , `corps2` , ... `corpsn`), lembre-se de que sua

URL será sempre http://localhost:8000, considerando que você vai reaproveitar a mesma porta para todos os projetos, parando e reiniciando o servidor conforme progride nos exercícios.

6.2 Edição de projeto

Aqui explicaremos o conteúdo das pastas e arquivos da raiz do projeto `corps1`. O arquivo `index.php` dentro da pasta `public` executa o trabalho básico de configurar a aplicação, executá-la e retornar a resposta.

A pasta `config` contém a configuração usada pelo componente `Laminas\ModuleManager` para carregar módulos e mesclar configurações. É possível separar a configuração por ambiente, e cada módulo possui a sua própria configuração.

A pasta `vendor` contém todos os módulos genéricos ou bibliotecas de terceiros. O conteúdo dessa pasta não deve ser modificado diretamente. Instalações e atualizações podem (e devem) ser feitas com o Composer.

A pasta `module` é onde criamos módulos específicos para a aplicação. Entretanto, devemos tentar escrevê-los para que sejam reutilizáveis. A carreira do módulo começa na pasta `module` e termina na pasta `vendor`, onde ele será gerenciado pelo Composer. Você pode publicar seu módulo em um repositório de onde o Composer pode instalá-lo e atualizá-lo.

Lembra que vimos sobre modularização no capítulo de introdução? O módulo é o cerne da aplicação Laminas e trabalharemos com módulos em todos os projetos de exemplo deste livro.

O esqueleto de projeto usado pelo Eclipse PDT vem com um módulo padrão, o Application. Se o módulo na rota não for especificado, a camada de controle assumirá que é ele graças a

uma rota pré-configurada. Quando o controlador de página não for especificado, será instanciada a classe `IndexController`, que nesse caso está no namespace `Application\Controller`.

A implementação da camada de controle de um módulo fica na subpasta `src`. A estrutura do módulo Application é mostrada na próxima figura.

Figura 6.4: Estrutura do módulo Application

Dentro do módulo padrão, encontramos as seguintes pastas:

- `config` : configuração do módulo;
- `language` : arquivos de tradução das mensagens do módulo;
- `srcv` : a implementação das camadas de back-end do módulo;

- `view` : a implementação de layouts e visões do módulo.

Na pasta `src` , também encontramos o arquivo `Module.php` , que contém a classe responsável pela configuração, inicialização e execução dos serviços do módulo. Na verdade, o arquivo `Module.php` é o mínimo requerido para um módulo. Isso significa que você pode substituir a implementação MVC do Laminas por outra.

6.3 Criação de módulo

Criaremos um novo módulo para conter os cadastros de setores espaciais e Lanternas Verdes. O novo módulo será chamado `Tropa` . Como o criaremos? Copiando o módulo `Application` .

Copie toda a pasta `Application` dentro de `module` , com o nome de "Tropa". Não é para copiar apenas o conteúdo, mas a pasta inteira. Edite o arquivo `Module.php` na pasta `src` , alterando o `namespace` :

```
<?php

namespace Application;
```

Para:

```
<?php

namespace Tropa;
```

Em seguida, edite o arquivo `IndexController.php` na pasta `src/Controller` , alterando o `namespace` :

```
<?php

namespace Application\Controller;
```

Para:

```php
<?php

namespace Tropa\Controller;
```

A partir de agora, você pode se defrontar com erros decorrentes da operação de cópia. Se algo não funcionar no módulo `Tropa`, verifique primeiro se o `namespace` de seus arquivos é `Tropa`, e não `Application`.

Cada módulo tem o seu `namespace`, que é o próprio nome do módulo. Quando copiamos o módulo `Application`, trouxemos uma classe controladora chamada `IndexController`, que é o nome do controlador padrão de um módulo. Mas não será esse o controlador com o qual trabalharemos agora. Criaremos outro, chamado `SetorController`, pois nossa primeira entrega será o cadastro de setores.

Copie o arquivo `IndexController.php` em `Tropa/src/Controller` como `SetorController.php`. Abra o arquivo copiado, e altere o seu conteúdo conforme a seguir:

SetorController.php

```php
<?php

namespace Tropa\Controller;

use Laminas\Mvc\Controller\AbstractActionController;
use Laminas\View\Model\ViewModel;

class SetorController extends AbstractActionController
{
    public function indexAction()
    {
        return new ViewModel();
    }
}
```

Considere que, exceto se houver referência em contrário, estaremos nos referindo ao módulo `Tropa`. Ciente disso, dentro da pasta `src`, crie as pastas `Form` e `Model`, que vão guardar classes que definem

formulários e modelos, respectivamente. Para criar uma nova pasta no Eclipse, selecione a pasta onde ela será contida e, com o botão direito do mouse, acesse o menu `New ? Folder`.

Não basta que a pasta do módulo exista dentro de `modules`. Para que `Laminas\ModuleManager` considere o módulo, ele precisa estar contido no array definido pelo arquivo `modules.config.php` em `corps1\config`. Verifique se o elemento está assim:

modules.config.php

```
return [
    'Laminas\Router',
    'Laminas\Validator',
    'Application',
];
```

Inclua o elemento `Tropa` nesse array, depois de `Application`. Se você esquecer disso, o roteamento de que trataremos adiante falhará miseravelmente. O conteúdo do arquivo `modules.config.php` ficará assim:

modules.config.php

```
return [
    'Laminas\Router',
    'Laminas\Validator',
    'Tropa',
    'Application',
];
```

Vamos deixar o módulo Application como último para que o layout deste seja usado como padrão. O último módulo da lista é o que o define o layout.

Agora podemos executar a aplicação e tentar acessar a URL http://localhost:8000/tropa? Não! Se tentarmos fazer isso, obteremos como resultado um erro 404. Acalme-se, pois ainda temos alguns passos a dar.

6.4 Criação das rotas

O controlador `SetorController` do módulo `Tropa` terá três ações:

- A que exibe a página inicial com a lista de setores;
- A que exibe o formulário de inclusão ou alteração do setor espacial;
- A que remove o setor espacial.

Os nomes das ações serão, respectivamente, `index` , `edit` e `delete` . Isto é, essas ações serão executadas pelas seguintes rotas:

- `corps1/tropa/setor`
- `corps1/tropa/setor/edit`
- `corps1/tropa/setor/delete`

Precisamos mapear essas ações, ou seja, relacionar o nome delas com as classes a serem instanciadas e os métodos a serem invocados. Isso é realizado no arquivo `module\Tropa\config\module.config.php` . A primeira providência é alterar o `namespace` do arquivo de:

module.config.php

```
namespace Application;
```

Para:

module.config.php

```
namespace Tropa;
```

Em seguida, vamos configurar determinadas chaves do array que ele contém. Primeiro, adicione uma referência à classe `SetorController` na chave `factories` de `controllers` , de modo que ela fique assim:

module.config.php=>controllers

```
    'controllers' => [
        'factories' => [
            Controller\IndexController::class => InvokableFactory::class,
            Controller\SetorController::class => InvokableFactory::class
        ],
    ],
```

A chave `factories` contém um array associativo que relaciona os namespaces dos controladores com classes que fabricam instâncias deles. Essa é uma diferença importante entre a versão 2 e 3 do Laminas.

No Laminas, é possível criar fábricas de controladores, o que permite injetar dependências neles, dotando cada uma das funcionalidades individuais necessárias sem gerar sobrecarga por meio de herança. Somente os controladores que estiverem presentes em `factories` serão criados.

Agora vamos configurar o roteamento do módulo. Não estamos trabalhando com acesso direto aos arquivos PHP, de modo que a requisição HTTP deve ser interpretada pelo Laminas para que uma classe que a trate seja instanciada.

A chave que contém a configuração de roteamento é `router`, ou, mais precisamente, a subchave `routes`. Altere a chave `home` para `tropa` e o conteúdo da subchave `route` de `/` para `/tropa`. A chave `router` deverá ficar exatamente como mostra o trecho de código a seguir, contendo apenas a rota `tropa`. Não deixe de remover a rota `application`, que foi copiada.

module.config.php=>router

```
    'router' => [
        'routes' => [
            'tropa' => [
                'type' => Segment::class,
                'options' => [
                    'route'    => '/tropa[/:controller[/:action]]',
                    'defaults' => [
```

```
                    'controller' => Controller\IndexController::class,
                    'action'     => 'index',
                ],
            ],
        ],
    ],
],
```

Preste atenção e pare de ficar vendo vídeos de gatinho. A chave `routes` é um conjunto de rotas, que são associações entre URLs e métodos de classes controladoras. As chaves das rotas não têm necessariamente ligação ao conteúdo da URL, embora seja claro que, se a chave for a raiz da URL, será muito mais fácil identificá-la. A chave da rota é utilizada por métodos que geram URLs.

Cada rota deve ter um tipo, que é uma classe do componente `Laminas\Router` que determina se a URL será interpretada ou não. O valor `Literal` é o tipo mais simples, quando não há interpretação da URL.

Cada rota tem um array `options`, que contém as informações de origem e destino – quer dizer, qual a URL e qual o método que será executado. Essas informações podem ser relativas ou absolutas, de acordo com o tipo de rota.

A opção `route` contém a URL ou o padrão de URL esperado. É o que virá na requisição após o nome da aplicação. Como o objeto roteador sabe separar a rota do nome da aplicação é algo que veremos no próximo capítulo.

O valor `Segment` para a chave `type`, que usamos na configuração das rotas do módulo `Tropa`, indica que ela especificará marcadores de posição que mapearão partes (segmentos) da rota para parâmetros específicos. Os dois pontos antes de um segmento da rota identificam-no como um parâmetro. Os segmentos são separados por barras, como se fossem níveis de diretório. Os colchetes determinam segmentos opcionais.

Isso significa que, ao invocar http://localhost:8000/tropa, a requisição será mapeada somente para o controlador `Tropa/Controller/IndexController`. Mas ao invocar `http://localhost:8000/tropa/[nome do controlador]`, a chave `controller` será alimentada pelo nome passado na URL. Quer dizer, existe um casamento exato na rota principal (denominada mãe) e um casamento baseado em um padrão na rota secundária (denominada filha).

Os argumentos básicos de um roteamento são o nome da classe controladora (controller) e o nome do método de resposta à requisição (action). É possível determinar regras para valores aceitáveis, na chave `options`, de modo que somente a presença do argumento não seja suficiente.

No trecho de código a seguir, vemos um exemplo de configuração de restrições, em que o argumento `action` admite somente letras, números e o sublinhado, enquanto o argumento `id` admite somente dígitos decimais. A chave `constraints` fica dentro da chave `options`.

module.config.php=>constraints

```
'constraints' => [
    'action' => '[a-zA-Z0-9_-]+',
    'id' => '[0-9]+',
],
```

É possível definir valores para serem assumidos na ausência de passagem dos argumentos com a chave `defaults`. É assim que vemos a página de boas-vindas do Laminas, chamada pelo método `indexAction()` da classe `Application\Controller\IndexController` sem que precisemos especificar qualquer um dos dois argumentos na URL.

O carregamento de classes em uma aplicação Laminas é delegado totalmente ao Composer. Isso implica na necessidade de configurar o carregamento de classes de cada módulo no arquivo

`composer.json`. Altere a chave `autoload` do arquivo `composer.json`, de modo que fique como mostra a listagem a seguir:

composer.json=>autoload

```
    "autoload": {
        "psr-4": {
            "Application\\": "module/Application/src/",
            "Tropa\\": "module/Tropa/src/"
        }
    },
```

Execute o comando `composer dumpautoload` para que o arquivo `autoload.php` seja alterado. O arquivo `autoload.php` faz a importação dos scripts que configuram o carregamento de classes do PHP pelo Composer. Se alguma das dependências do projeto possui uma nova versão e a expressão no arquivo `composer.json` instrui o Composer a trazer a mais recente, neste momento também ocorrerá a atualização dessa dependência.

Renomeie a subpasta `application` em `view`, que foi copiada do módulo `Application`, para `tropa`. Na pasta `index` de `view` no módulo `Tropa`, edite o arquivo `index.phtml` e substitua o conteúdo por este:

view\tropa\index\index.phtml

```
<?php
echo __FILE__;
```

Neste momento, se você invocar a URL http://localhost:8000/tropa (ou http://localhost:8000/tropa/), verá uma página como a da figura a seguir. Ela indica que o template de página que alteramos foi realmente usado.

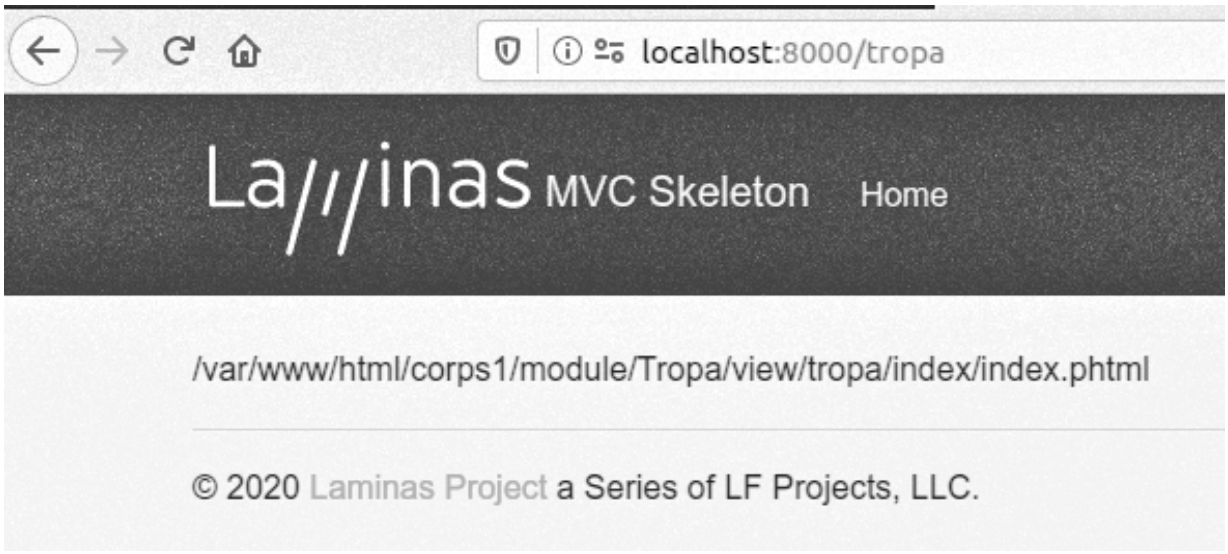


Figura 6.5: Página inicial de um módulo Tropa

Apague a pasta `layout` dentro de `view`, pois o módulo `Application` já define o layout.

Criando uma view para setor

Agora, se você invocar a URL http://localhost:8000/tropa/setor/, verá a página de erro da figura a seguir. O erro ocorre porque não existe a página HTML requerida pelo método chamado. De acordo com o roteamento configurado, o método chamado é `Tropa\SetorController::indexAction`.

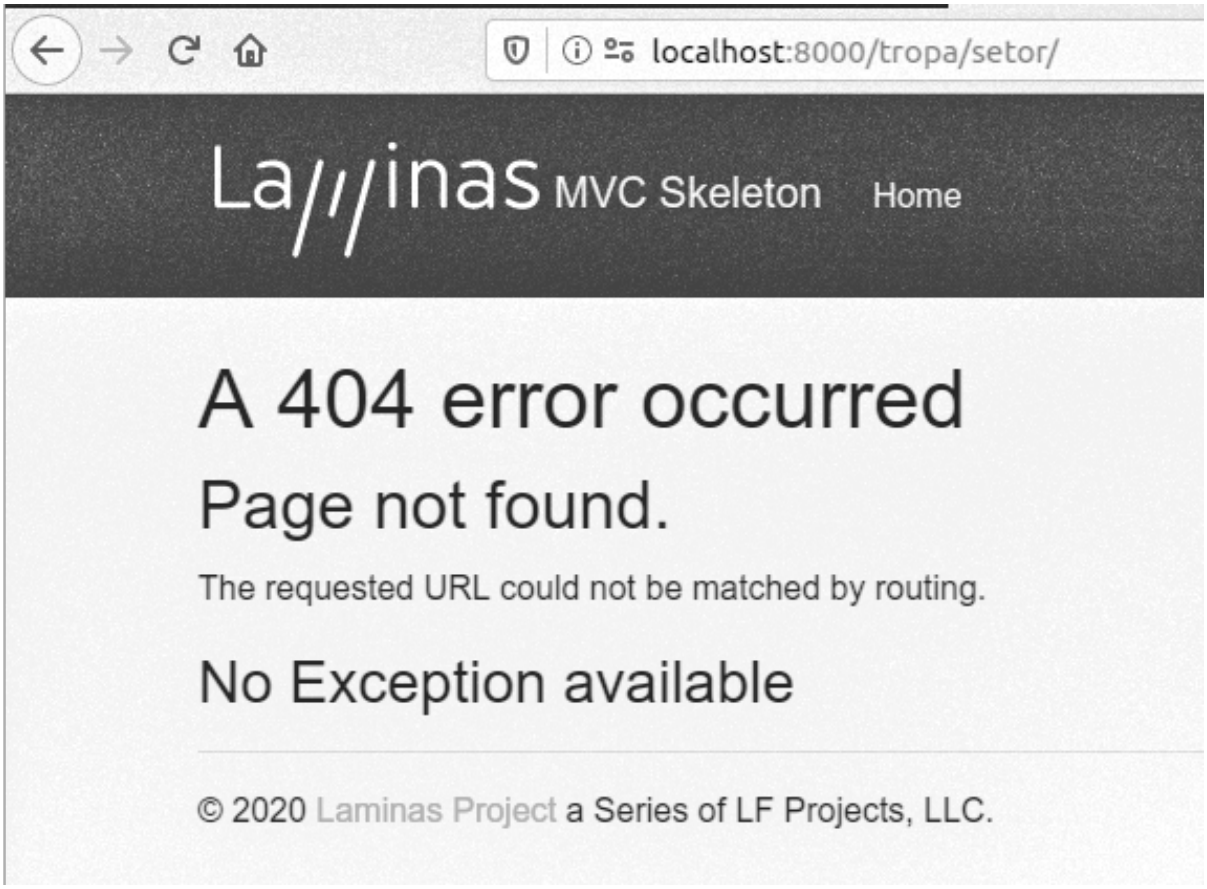

localhost:8000/tropa/setor/
Laminas MVC Skeleton
Home
A 404 error occurred
Page not found.
The requested URL could not be matched by routing.
No Exception available
© 2020 Laminas Project a Series of LF Projects, LLC.

Figura 6.6: Página de erro com exceção lançada

Exceto se houver um redirecionamento ou uma interrupção do processamento, o comportamento padrão de um controlador no Laminas é renderizar um arquivo HTML cujo nome corresponda ao método invocado sem o sufixo `Action`. Isso quer dizer que o método `indexAction` requer um arquivo `index.phtml` (essa é a extensão padrão das páginas do Laminas).

As páginas de um módulo ficam na pasta `view`, no mesmo nível da pasta `src`. Dentro da pasta `view`, deve haver uma pasta com o nome do módulo e, dentro desta, deve haver uma para cada controlador. Enquanto as subpastas de `src` seguem o padrão de nomes de classe da PSR-4, as subpastas de `view` são todas com letras minúsculas.

Seguindo esse padrão, precisamos criar o arquivo `index.phtml`, que representa a página inicial do controlador `SetorController`, na pasta `view/tropa/setor`. A pasta `setor` não existe, por isso crie-a, usando o menu `File ? New ? Folder`. Dentro dessa nova pasta, crie o arquivo `index.phtml` usando o menu `File ? New ? PHP File`. Edite-o e substitua seu conteúdo por este:

view\tropa\setor\index.phtml

```
<?php
echo __FILE__;
```

Em seguida, edite o arquivo `module.config.php` e adicione a chave `aliases` dentro de controllers, conforme a próxima listagem:

module.config.php=>controllers

```
    'controllers' => [
        'aliases' => [
            'setor' => Controller\SetorController::class
        ],
        'factories' => [
            Controller\IndexController::class => InvokableFactory::class,
            Controller\SetorController::class => InvokableFactory::class
        ],
    ],
```

Essa mudança é necessária para que o roteador saiba como chegar da palavra ‘setor’ no argumento ‘controller’ ao controlador `Tropa\Controller\SetorController`. Se você está se perguntando por que não é necessário configurar um apelido para o `IndexController`, lembre-se de que o valor padrão do argumento `controller` já é o namespace completo.

No arquivo `module.config.php`, há uma seção referente ao gerenciamento da interface com o usuário. Seu conteúdo deve ser o seguinte:

module.config.php=>view_manager

```
    'view_manager' => [
        'display_not_found_reason' => true,
        'display_exceptions'       => true,
        'doctype'                  => 'HTML5',
        'not_found_template'       => 'error/404',
        'exception_template'       => 'error/index',
        'template_map' => [
            'layout/layout'           => __DIR__ .
'/../view/layout/layout.phtml',
            'application/index/index' => __DIR__ .
'/../view/application/index/index.phtml',
            'error/404'               => __DIR__ .
'/../view/error/404.phtml',
            'error/index'             => __DIR__ .
'/../view/error/index.phtml',
        ],
        'template_path_stack' => [
            __DIR__ . '/../view',
        ],
    ],
```

Para que a camada de visão do módulo `Tropa` funcione perfeitamente, altere a chave `application/index/index` de `template_map` , para que fique assim:

```
            'tropa/index/index' => __DIR__ .
'/../view/tropa/index/index.phtml',
```

Essa troca é necessária porque o valor anterior veio de brinde com a cópia e refere-se aos caminhos do módulo `Application` .

Agora você pode invocar a URL http://localhost:8000/tropa/setor/ e ter uma página com o nome do arquivo de template, sem a mensagem de erro. Tenha a certeza de compreender o procedimento de criação da página associada ao método do controlador – que doravante chamaremos de visão –, pois implementa a responsabilidade de interface com o usuário do MVC.

6.5 Criação das ações do controlador

É preciso implementar os métodos que permitirão a alteração e remoção de registros do cadastro que estamos criando. Para isso, modifique a classe `SetorController`, acrescentando três métodos, como mostra a listagem a seguir: `editAction()`, `saveAction()` e `deleteAction()`.

SetorController

```
<?php
namespace Tropa\Controller;

use Laminas\Mvc\Controller\AbstractActionController;
use Laminas\View\Model\ViewModel;

class SetorController extends AbstractActionController
{
    public function indexAction()
    {
        return new ViewModel();
    }

    /**
     * Action to add and change records
     */
    public function editAction()
    {

    }

    /**
     * Action to save a record
     */
    public function saveAction()
    {

    }

    /**
```

```
     * Action to remove records
     */
    public function deleteAction()
    {

    }

}
```

O método `indexAction()` veio de brinde quando criamos o controlador anteriormente. Observe que ele retorna um objeto da classe `ViewModel`. O que seria isso?

Vimos na introdução que o MVC tem Model, View e Controller, mas um `ViewModel` pertenceria a qual camada? A nenhuma do MVC, porque `ViewModel` faz parte de outro padrão, o MVVM (*Model-View-ViewModel*) – criado por John Gossman em 2005.

Segundo Melia (2013, p. 2), "enquanto uma Visão pode ser ligada aos dados no Modelo ou às ações contra esses dados, na prática, isso não é feito diretamente, mas por outra 'camada' chamada ViewModel ou VM". Melia afirma que ViewModel é um Modelo da Visão e define isso como algo como uma abstração da Visão que, ao mesmo tempo, é uma especialização do Modelo com o qual a Visão pode se ligar.

O que Melia (2013) quer dizer é o seguinte: no mundo perfeito, a camada View tem acesso direto aos dados da camada Model. Entretanto, no projeto Laminas, a camada View não tem acesso direto aos dados da camada Model. Os dados da camada View são passados pela camada Control, que exerce de forma efetiva a sua responsabilidade de controlar o fluxo de dados. A camada ViewModel é o meio de transporte que a camada Control usa para enviar os dados da camada Model para a camada View.

6.6 Criação de layout

Cada módulo tem um layout próprio. Por padrão, ele fica no diretório `view\layout` no arquivo `layout.phtml`. Vamos editar o `layout.phtml` do módulo `Tropa`, preenchendo-o com um conteúdo mínimo, que expandiremos gradualmente.

Como o módulo `Tropa` foi copiado do módulo `Application`, a pasta `layout` já existe dentro de `view` e, dentro dela, já existe o arquivo `layout.phtml`. O conteúdo desse arquivo não nos serve, pois nossa aplicação terá um layout diferente.

Vamos alterar o arquivo `layout.phtml` explicando o seu novo conteúdo passo a passo. O passo inicial é apagar todo o conteúdo atual do arquivo `layout.phtml`. Com o arquivo limpo, vamos acrescentar os trechos a seguir. Primeiro, adicione as seguintes linhas:

```
<?=$this->doctype()?>
```

O método `doctype()` imprime a tag de tipo de documento de acordo com o valor de doctype configurado dentro das opções da chave `view_manager` do arquivo `module.config.php`. Na ausência de definição, será impresso o doctype de HTML 5. Observe que se você abrir o arquivo `module.config.php` do módulo `Tropa`, verá um exemplo de declaração de doctype.

Em seguida, adicione estas linhas:

```
<html lang="en">
<head>
<meta charset="utf-8">
        <?=$this->headTitle('Sistema de Controle da Tropa dos Lanternas
Verdes')?>
```

O método `headTitle()` gera uma tag `TITLE` de HTML, com o conteúdo especificado pelo argumento passado. Em seguida, adicione as próximas linhas:

```
        <!-- Styles -->
        <?=$this->headLink(array('rel' => 'shortcut icon','type' =>
'image/vnd.microsoft.icon','href' => $this->basePath() .
'/img/favicon.ico'))->prependStylesheet($this->basePath() .
'/css/tropa.css')?>
```

O método `headLink()` cria uma tag `LINK` de HTML. O array passado como argumento contém os valores dos atributos dessa tag. O método `prependStylesheet()` adiciona a referência ao arquivo CSS passado como argumento.

Em seguida, acrescente as linhas a seguir:

```
        <!-- Scripts -->
        <?=$this->headScript()->prependFile($this->basePath() .
'/js/jquery-3.5.1.min.js', 'text/javascript')?>
        <?=$this->headScript()->prependFile($this->basePath() .
'/js/bootstrap.min.js', 'text/javascript')?>
    </head>
```

O método `headScript()` gera uma tag `SCRIPT` de HTML, com o conteúdo do arquivo especificado como argumento no método `prependFile()`. Já os `headScript()`, `headLink()` e `headTitle()` fazem parte de uma família de métodos para geração dinâmica de conteúdo HTML.

Em seguida, adicione as seguintes linhas, agora referentes ao corpo do documento HTML:

```
<body>
    <header>
        <h1>Sistema de Controle da Tropa dos Lanternas Verdes</h1>
    </header>
    <div class="content">
        <?=$this->content?>
    </div>
```

O atributo `content` tem como valor o conteúdo do arquivo `.phtml` correspondente ao método do controlador que respondeu à requisição atual.

Finalmente, complete o arquivo com as seguintes linhas:

```
    <footer>
        <p>&copy; <?=date('Y') ?> by Oa Sector 0 <?php echo $this-
>translate('All rights reserved.') ?></p>
    </footer>
</body>
</html>
```

O método `translate()` retorna a tradução do texto passado como argumento, de acordo com o idioma configurado como padrão para a aplicação.

Como você pode notar, esse arquivo de layout importa uma folha de estilo que não existe no projeto. Portanto, vamos criá-la. Entre na pasta `public/css` e crie o arquivo `tropa.css` . Você pode usar o menu `File ? New ? CSS File` para fazer isso.

public\css\tropa.css

```
@CHARSET "UTF-8";

header {
    background-color: white;
    border-color: black;
    border-style: solid;
    color: darkgreen;
    font-family: verdana;
    font-weight: bold;
    padding: 10px;
    text-align: center;
    width: 98%;
}

body {
    background-color: darkmagenta;
    color: white;
}

body a{
    color: white;
```

```
}

footer {
    background-color: red;
    border-color: green;
    border-style: solid;
    bottom: 0%;
    color: white;
    font-family: verdana;
    font-weight: bold;
    text-align: center;
    font-weight: bold;
    position: absolute;
    width: 98%;
}
```

Você se lembra da chave `modules` do array retornado pelo arquivo `application.config.php` ? Aquela que relaciona os módulos ativos na aplicação? Pois bem, a ordem dos nomes dos módulos na chave importa para a determinação do layout da aplicação.

Embora cada módulo possa definir um layout, o layout que será considerado e aplicado para todos será o do último módulo definido no array `modules` . Para que tenhamos como resultado a página inicial do cadastro de setores ao recarregar a página para a URL http://localhost:8000/tropa/setor/, precisamos garantir que o módulo `Tropa` seja o último elemento do array `modules` . Se você não alterar a ordem, mantendo o módulo `Application` como último elemento, verá que o layout permanece o padrão do esqueleto de aplicação.

Mas antes de recarregar a página, vamos adicionar uma dependência ao nosso projeto, para que não ocorra uma exceção devido à falta da classe auxiliar de tradução. O componente de tradução do Laminas para aplicações que usam MVC é o `Laminas\Mvc\I18n` . Edite o arquivo `composer.json` e acrescente a dependência ao componente `Laminas\Mvc\I18n` , conforme mostra a listagem a seguir. Aproveitaremos e acrescentaremos o componente para testes unitários, o `Laminas\Test` :

composer.json=>require

```
    "require" : {
        "php" : "^7.3",
        "laminas/laminas-component-installer" : "^1.0 || ^2.1",
        "laminas/laminas-development-mode" : "^3.2",
        "laminas/laminas-skeleton-installer" : "^0.1.7 || ^1.0",
        "laminas/laminas-mvc" : "^3.1.1",
        "laminas/laminas-mvc-i18n": "^1.1",
        "laminas/laminas-test" : "^3.4"
    },
```

Execute o comando `composer update` e selecione sempre a opção padrão (default) quando questionado. A opção padrão de instalação de componentes do Laminas é injetar a configuração no arquivo `modules.config.php`, tratando o componente como um módulo da aplicação.

O componente `Laminas\Mvc\I18n` – componente de tradução para módulos de aplicações Laminas – depende de outro, o `Laminas\I18n`, que é um componente de tradução genérico para aplicações PHP. Se você tivesse concordado em injetar esses componentes quando o Composer lhe questionou, ele os teria incluído no arquivo `modules.config.php`.

Como você não fez isso, porque nós queremos que saiba o que ocorre atrás das cortinas, abra o arquivo `modules.config.php` e adicione os `Laminas\Mvc\I18n` e `Laminas\I18n` de modo que o array do arquivo fique assim:

modules.config.php

```
return [
    'Laminas\Mvc\I18n',
    'Laminas\I18n',
    'Laminas\Router',
    'Laminas\Validator',
    'Application',
    'Tropa',
];
```

Figura 6.7: Layout do módulo Tropa

Lembre-se de que, no caso do layout, o último módulo declarado tem preferência no layout. Cada módulo pode definir um layout, mas somente um será usado por toda a aplicação – exceto se um evento ou o controlador modificá-lo de forma explícita.

Vamos testar a chamada à ação de exibição do formulário de edição de nosso cadastro. Mas, para isso, é necessário que a respectiva página exista, mesmo que esteja vazia. Logo, vamos encerrar esta seção criando mais um arquivo PHP vazio com o nome de `edit.phtml`. Você o criará na pasta `module\Tropa\view\tropa\setor`. Use o menu `File ? New ? PHP File`.

O que deve ocorrer agora? Você deve ser capaz, a esta altura, de conseguir invocar a URL http://localhost:8000/tropa/setor/edit. Se tudo correu bem, você verá apenas o layout sem conteúdo.

No próximo capítulo, vamos implementar a conexão com o banco de dados, e a leitura e a gravação de dados em tabelas. A próxima

seção é uma leitura adicional que mostra uma alternativa para a criação de projetos PHP no Eclipse com o Composer.

6.7 Integração do Eclipse e do Composer

Nesta seção, mostraremos que qualquer ferramenta que pode ser executada pelo terminal também pode ser executada pelo Eclipse, como se fizesse parte dele e com a automatização das partes genéricas dos comandos executados via terminal. Isso resulta em mais produtividade por menos digitação e menos alternância entre janelas.

Veremos aqui como a criação do projeto `corps1` poderia ser feita com a configuração de execução de programas externos.

Abra o Eclipse PDT, e certifique-se de que o workspace selecionado é o diretório `webroot` . Abra o menu `Run ? External Tools ? External Tools Configuration` . De acordo com a próxima figura, crie um novo item clicando duas vezes sobre o nó `Program` no quadro da esquerda. Preencha o formulário da aba `Main` com os seguintes valores:

Campo	Valor
Name:	`composer create project`
Location:	[DIRETÓRIO ONDE O COMPOSER FOI INSTALADO]/composer
Working Directory:	`${workspace_loc}`
Arguments:	`create-project -sdev laminas/laminas-mvc-skeleton ${string_prompt:project name}`

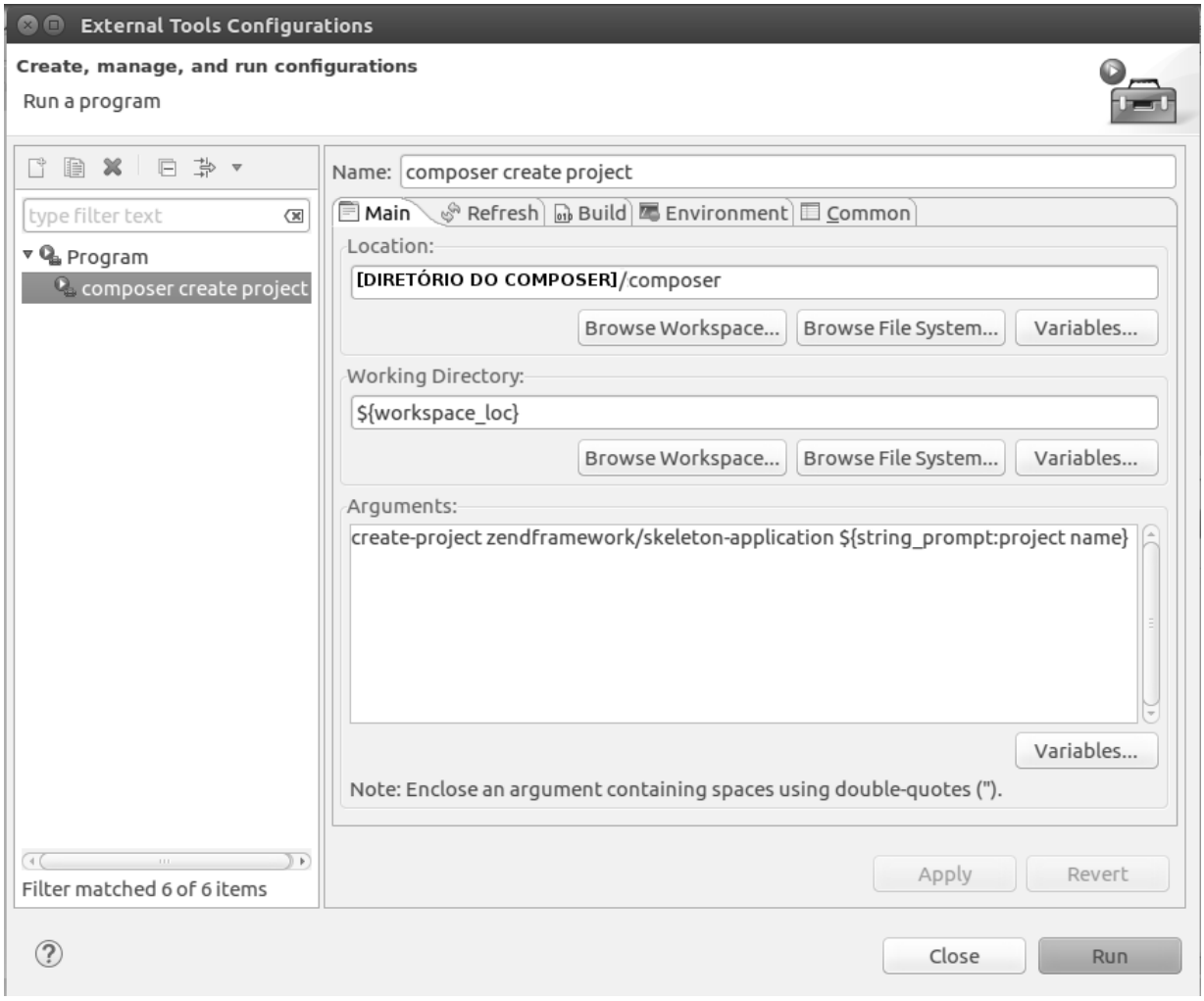


Figura 6.8: Configuração de chamada ao criador de projetos do Composer no Eclipse

Conforme mostra a figura a seguir, na aba `Common`, no quadro `Display in favorites menu`, selecione o item `External Tools`, para que o comando criado fique visível na barra de ferramentas. Clique no botão `Apply` para gravar a configuração e em `Close` para fechar a janela.

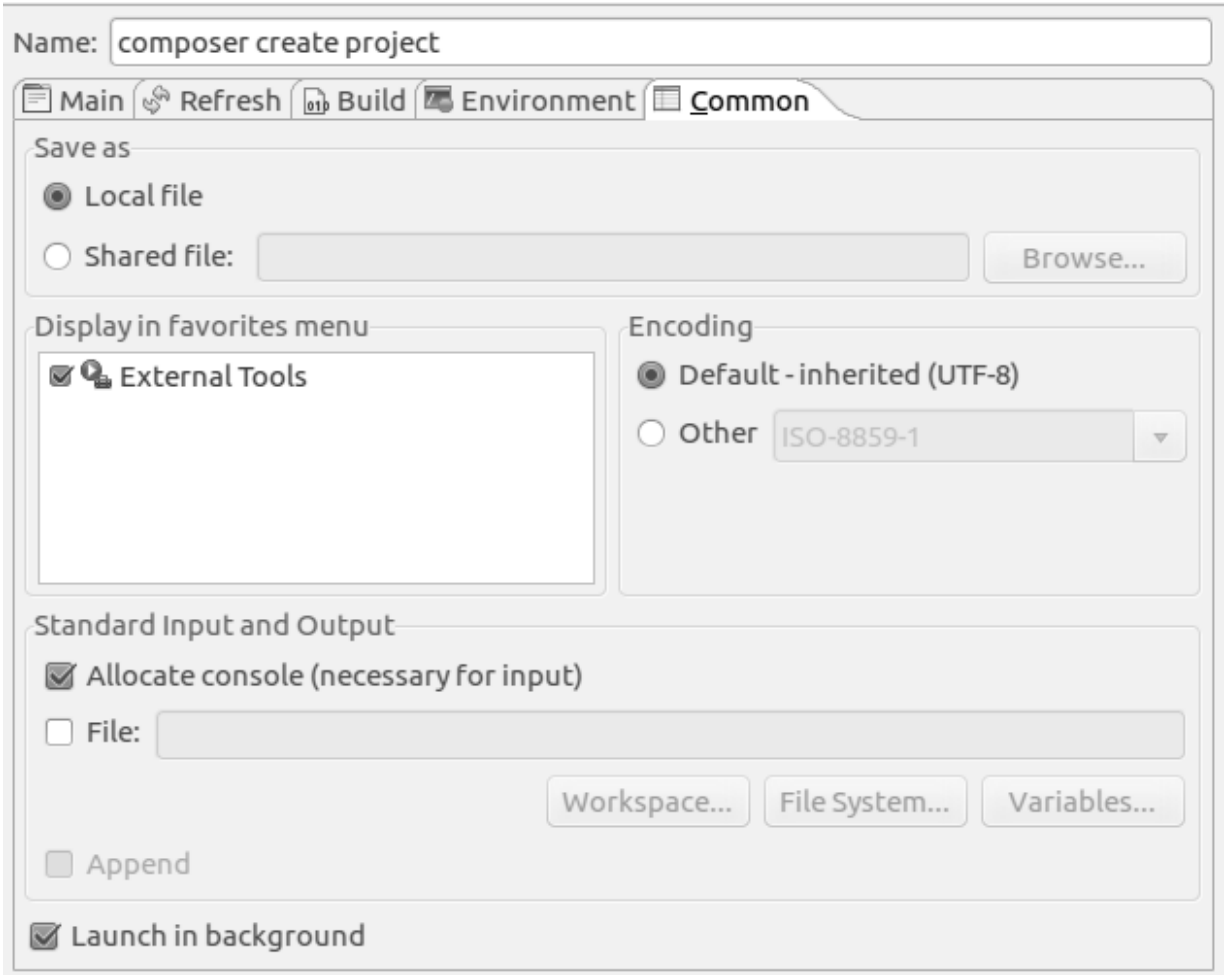


Figura 6.9: Habilitação da visibilidade para o comando de criação de projetos com Composer

Para criar o projeto a partir da barra de ferramentas do Eclipse, clique sobre o botão `Run External Tools`, o que tem um disco verde com uma seta branca e uma maleta vermelha. Ele abrirá uma lista, onde você verá o item `composer create project`, conforme a próxima figura.

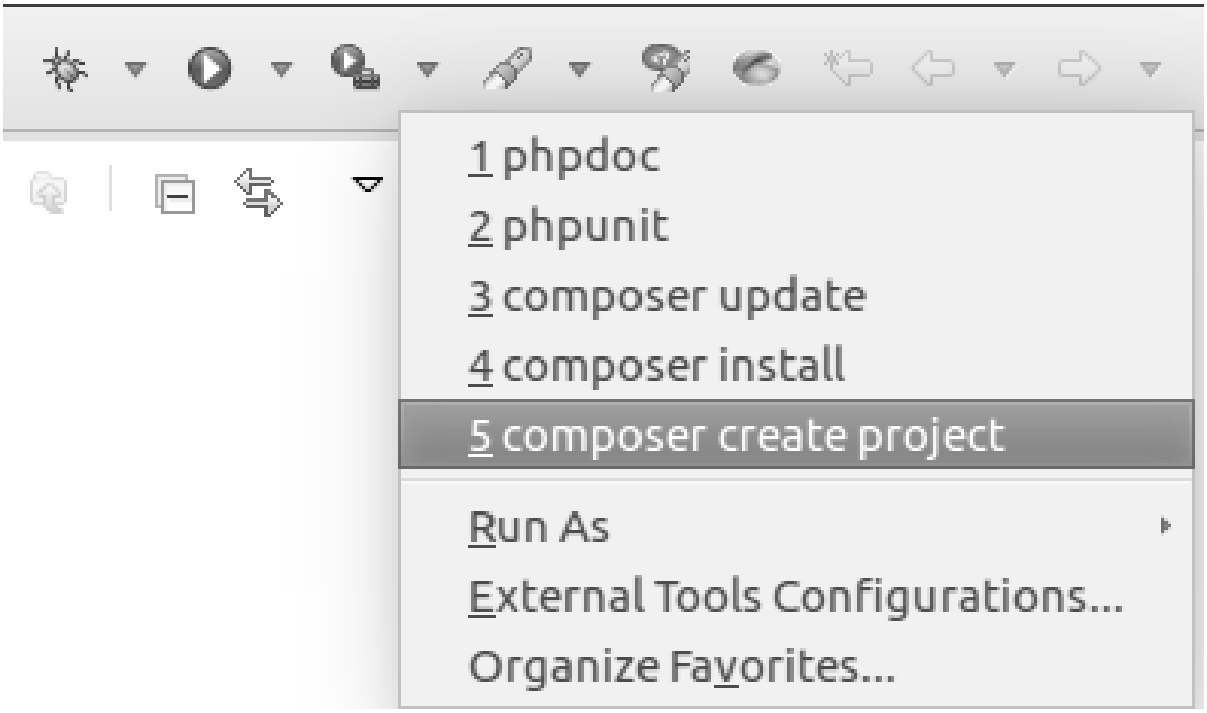


Figura 6.10: Comando de criação de projetos com Composer na barra de ferramentas

Se o botão `Run External Tools` não estiver visível, você deve acessar o menu `Window ? Customize Perspective` . Na aba `Tool Bar Visibility` , abra o item `Launch` e marque o subitem `External Tools` , de acordo com a figura a seguir. Após clicar em `OK` , e o botão vai aparecer na barra de ferramentas.

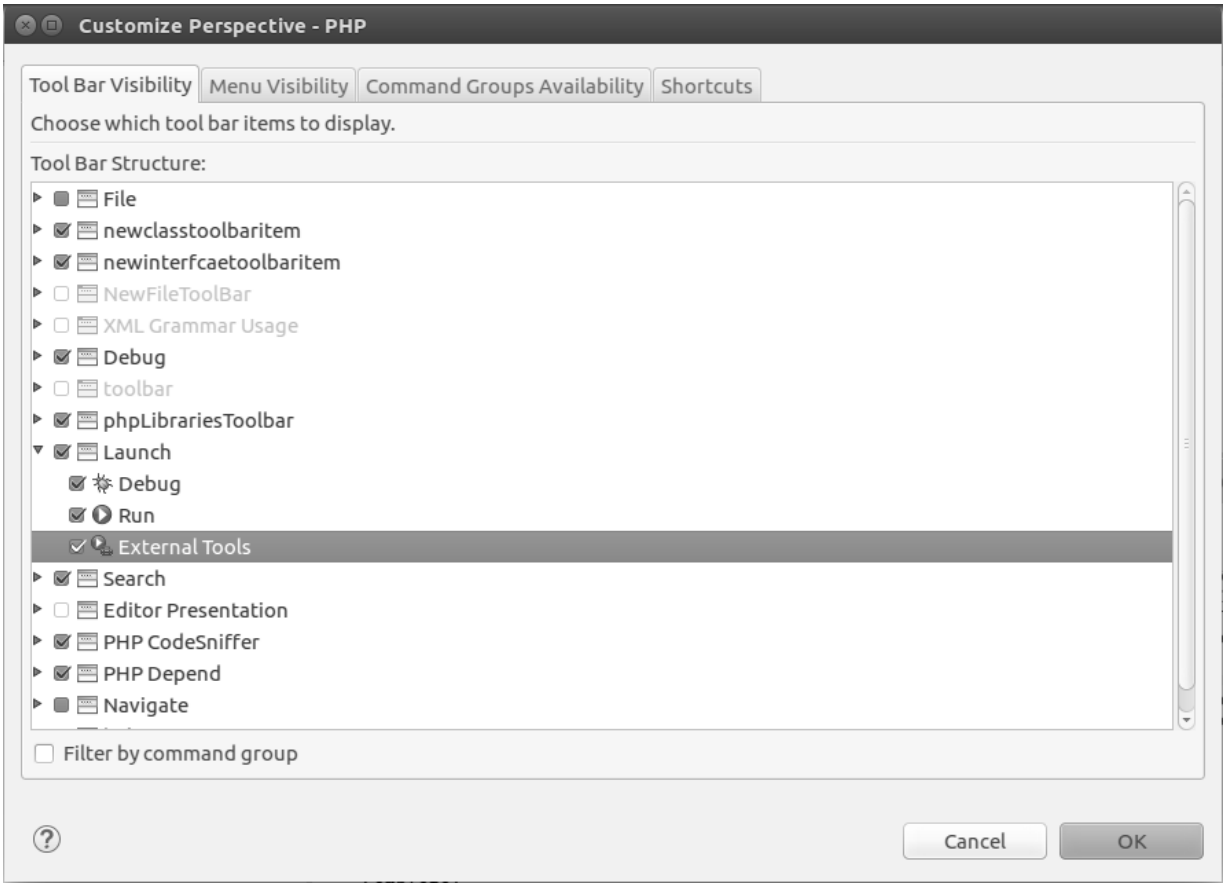


Figura 6.11: Habilitação do botão Run External Tools

Após clicar sobre o item `composer create project`, será aberta uma caixa de diálogo, conforme a próxima figura, na qual você digitará o nome do projeto (`corps1`). Após clicar no botão `OK`, o Composer vai mostrar quais pacotes de software estão sendo baixados e quais são as suas respectivas versões. Entretanto, isso não vai ocorrer se o projeto `corps1` já existir.

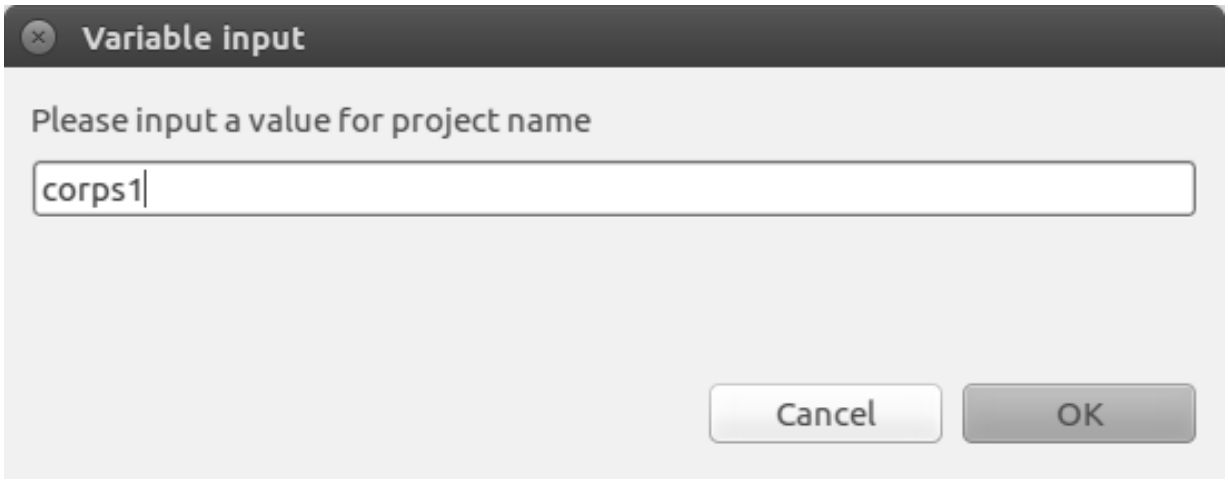


Figura 6.12: Janela para informar o nome do projeto

Capítulo 7
Mapeamento objeto-relacional

Embora existam várias técnicas para embutir SQL em uma linguagem de programação, elas são todas um pouco deselegantes.
– Martin Fowler

Neste capítulo, prosseguiremos na construção do projeto iniciado no capítulo anterior. Aprenderemos a estabelecer uma conexão entre uma aplicação orientada a objetos e um banco de dados relacional, a criar uma classe que mantenha em memória os dados armazenados em disco pelo banco de dados e outra classe que faça o transporte dos dados entre a memória e o disco.

7.1 Banco de dados, modelo e mapeador

Sem dados, não podemos fazer muita coisa, incluindo ganhar dinheiro em algum cassino de Las Vegas. Por isso, vamos criar o banco de dados que será utilizado pela aplicação, pois as tabelas desse banco serão mapeadas para as classes da camada de Modelo de nossa aplicação. E nossa jornada rumo ao diamante começa com o pedaço de carvão.

Acesse o *phpMyAdmin* por meio da URL http://localhost/phpmyadmin. Claro, isso se você instalou o phpMyAdmin ou o XAMPP, do qual falamos no capítulo sobre ambiente de desenvolvimento. Se você tiver como resultado uma página de erro 403, então utilize o programa ou a página de segurança do XAMPP para configurar uma senha para o phpMyAdmin. Se você chamar o executável do XAMPP no terminal com o parâmetro `security`, ele iniciará um modo iterativo que perguntará quais itens de segurança você deseja habilitar.

Ao acessar a URL citado anteriormente, será apresentada a página de autenticação. Após a autenticação, você será encaminhado para a página principal do phpMyAdmin. Esses dois passos são ilustrados nas figuras a seguir.

Figura 7.1: Página de autenticação do phpMyAdmin

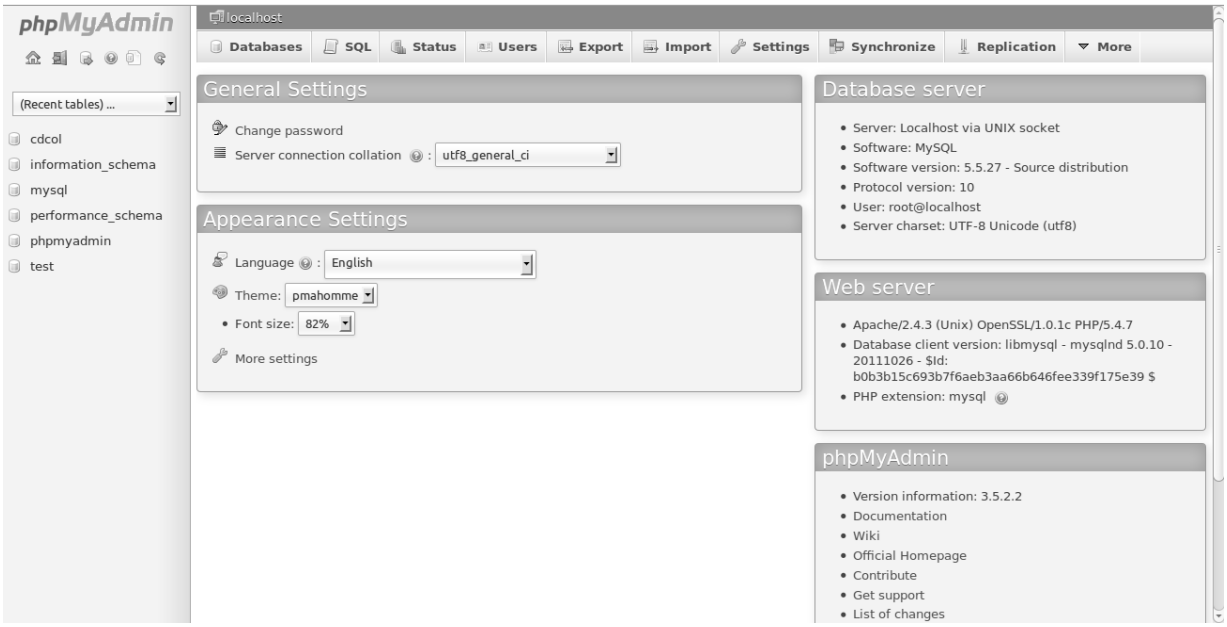


Figura 7.2: Página principal do phpMyAdmin

Uma vez autenticado, clique na aba `Databases`, na parte superior esquerda do quadro direito. Será apresentada a página `Databases` conforme a próxima figura. Preencha o campo `Create database` com o nome `corps` e clique no botão `Create`.

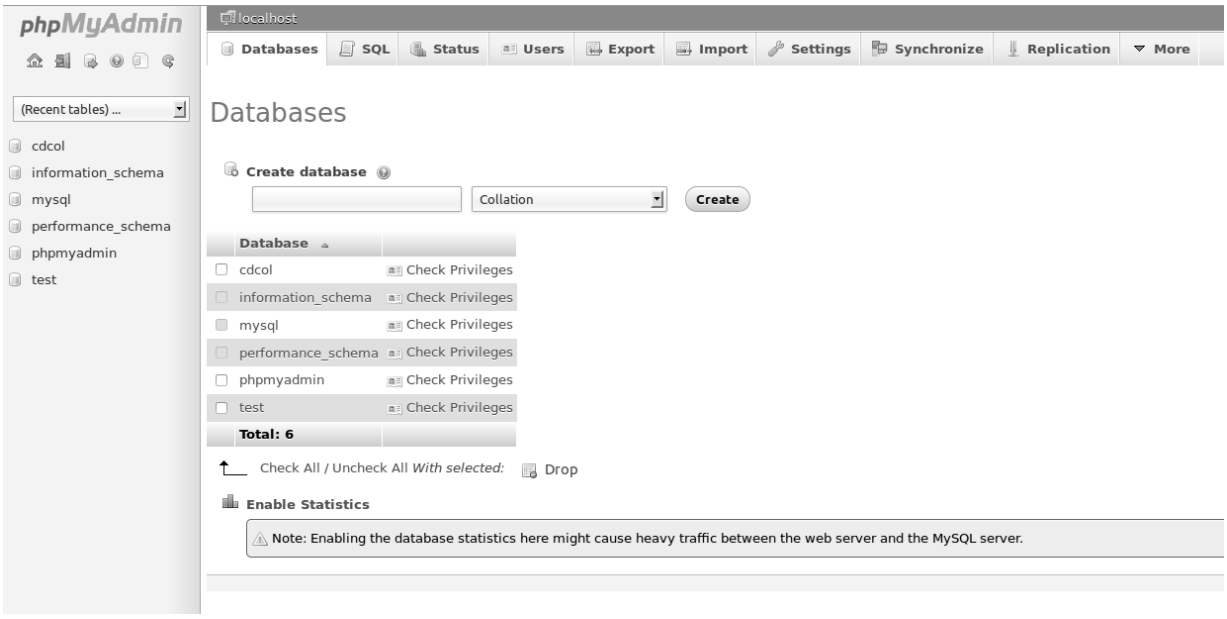


Figura 7.3: Página Databases do phpMyAdmin

A página será atualizada e exibirá o banco de dados `corps` na tabela junto aos demais. Clique sobre o nome do banco `corps` e os quadros serão atualizados para as páginas de criação de tabelas, conforme se vê na figura a seguir.

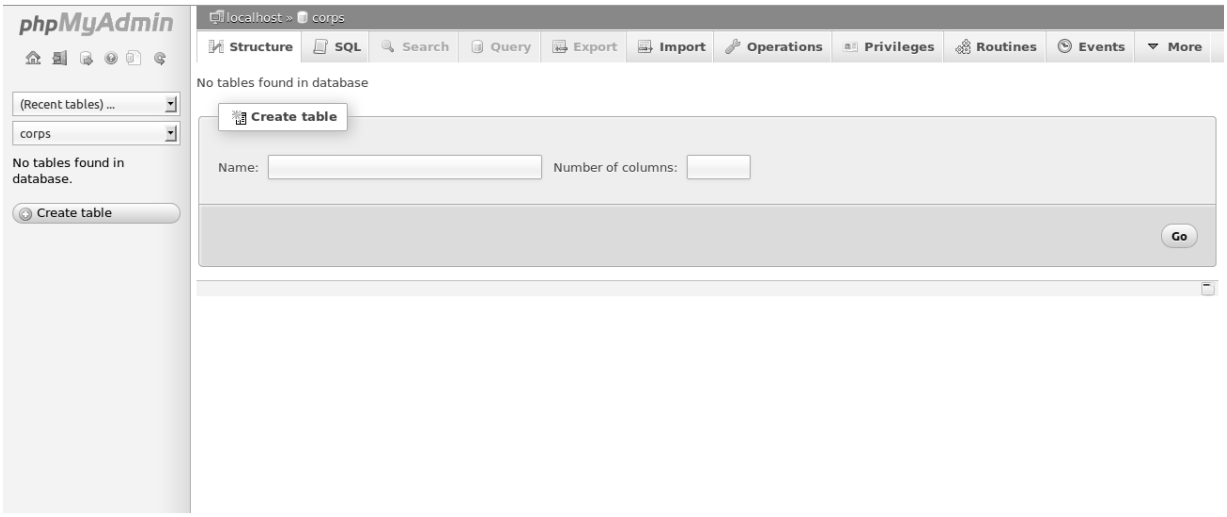


Figura 7.4: Páginas de criação de tabelas do phpMyAdmin

Clique na aba `SQL` , que é a segunda da esquerda para a direita no quadro da direita. Será apresentada uma página (próxima figura) com o quadro onde podemos colocar instruções SQL para serem executadas sobre o banco selecionado.

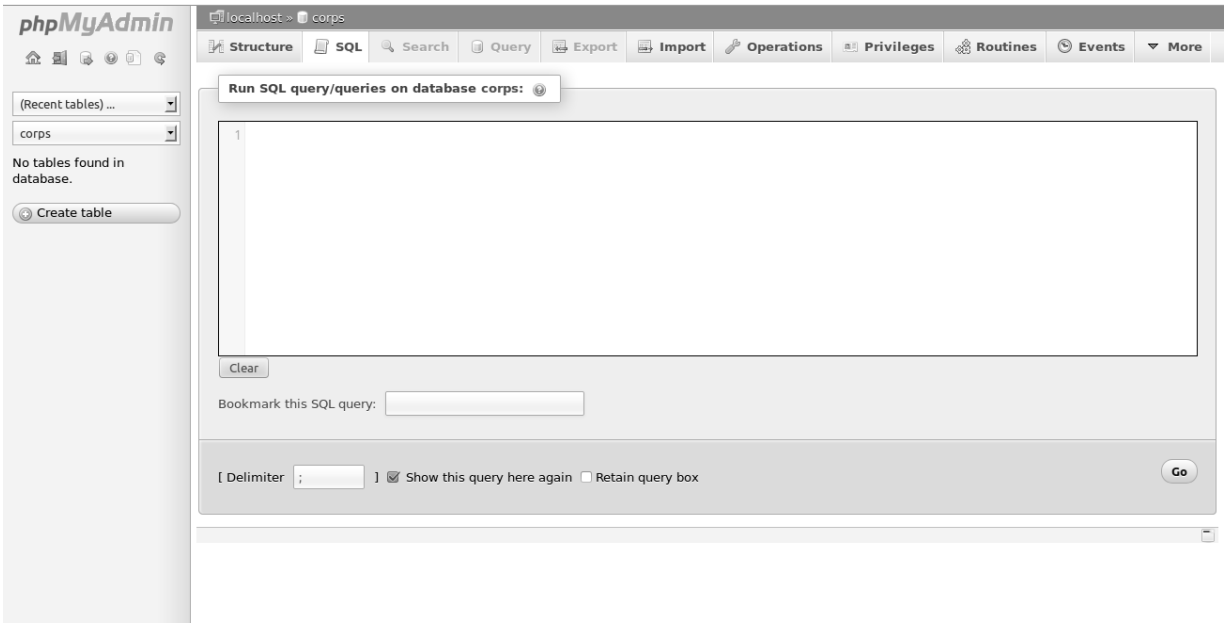


Figura 7.5: Página de execução de SQL do phpMyAdmin

Conforme falamos anteriormente, uma parte do negócio do nosso sistema envolve o armazenamento de dados de setores espaciais. Para isso, precisamos de uma tabela de setores. Vamos criá-la usando o comando `CREATE TABLE` de SQL. Insira a seguinte instrução SQL na área de texto:

```
CREATE TABLE setor(codigo int(11) NOT NULL, nome varchar(30) NOT NULL,
PRIMARY KEY (codigo));
```

Clique no botão `Go` e a tabela será criada.

> Se você adora o terminal, pode executar o cliente do MySQL do XAMPP e criar a tabela pelo modo texto em vez de usar a web. No Linux e no MacOS, os programas encontram-se na pasta `bin` do XAMPP. No Windows, o MySQL está na pasta `mysql`.

Agora, o momento que todos esperavam: a criação do modelo. Na pasta `module\Tropa\src\Model`, crie a classe `Setor`, com o namespace `Tropa\Model`. Uma classe PHP aderente a PSR-4 fica em um arquivo

com o mesmo nome. Você pode usar o menu `File ? New ? Class` para criar uma classe PHP.

O arquivo `Setor.php` , uma vez criado, deverá ter o seguinte conteúdo:

Setor.php

```php
<?php
namespace Tropa\Model;

class Setor
{

    /**
     *
     * @var integer
     */
    public $codigo;

    /**
     *
     * @var string
     */
    public $nome;

    /**
     *
     * @param array $data
     */
    public function exchangeArray(array $data)
    {
        foreach($data as $attribute => $value){
            $this->$attribute = $value;
        }
    }
}
```

Perceba que a classe que chamamos de modelo é uma entidade do sistema, cujos atributos correspondem, neste caso, pelo menos, diretamente aos campos da tabela do banco que vai manter os

dados. O método `exchangeArray()` será usado para que o objeto da classe `Setor` receba os dados de um registro da tabela `setor`.

Mas os dados de um registro da tabela `setor` não serão transportados para um objeto da classe `Setor` por magia. Eles serão transportados por um objeto de uma classe auxiliar, denominada *mapeadora*, que criaremos a seguir.

Crie o arquivo `SetorTable.php` em `module\Tropa\src\Model`, e comece adicionando as seguintes linhas:

```
<?php
namespace Tropa\Model;

use Laminas\Db\TableGateway\TableGatewayInterface;
use Laminas\Db\Adapter\Driver\ResultInterface;
```

A interface `TableGatewayInterface` define um padrão de métodos para manipulação de tabelas como objetos. Os métodos são correspondentes aos comandos da sublinguagem DML da SQL: `insert()`, `update()`, `delete()` e `select()`.

A interface `ResultInterface` define um padrão para encapsulamento dos resultados de uma consulta a tabelas de um banco de dados relacional em um objeto. Esse objeto é uma coleção iterável de itens. Em seguida, adicione as linhas a seguir:

```
class SetorTable
{
    /**
     *
     * @var TableGatewayInterface
     */
    private $tableGateway;

    /**
     *
     * @var string
     */
    private $keyName = 'codigo';
```

O atributo `tableGateway` vai armazenar um objeto de uma classe que implementará a interface `TableGatewayInterface`. O atributo `keyName` armazena o nome do campo que é chave primária na tabela mapeada pela classe.

Em seguida, acrescente as próximas linhas:

```
/**
 *
 * @param TableGatewayInterface $tableGateway
 */
public function __construct(TableGatewayInterface $tableGateway)
{
    $this->tableGateway = $tableGateway;
}
```

O método construtor da classe vai receber obrigatoriamente como argumento um objeto que implementa a interface `TableGatewayInterface`, garantindo que o atributo `tableGateway` contenha esse objeto. Esse é um exemplo de injeção de dependência via construtor. A classe `SetorTable` depende da injeção de uma implementação de `TableGatewayInterface`.

Em seguida, adicione as próximas linhas:

```
/**
 *
 * @return ResultInterface
 */
public function fetchAll()
{
    $resultSet = $this->tableGateway->select();
    return $resultSet;
}
```

O método `fetchAll()` é uma fachada para o método `select()` da implementação de `TableGatewayInterface`. Esse método, executado sem argumentos, vai fazer uma consulta do tipo `SELECT * FROM [NOME DA TABELA]`.

Em seguida, acrescente as próximas linhas:

```
    /**
     *
     * @param string $keyValue
     * @return Setor
     */
    public function getModel($keyValue)
    {
        $rowset = $this->tableGateway->select(array(
            $this->keyName => $keyValue
        ));
        if ($rowset->count()>0){
            $row = $rowset->current();
        } else {
            $row = new Setor();
        }

        return $row;
    }
```

O método `getModel()` retornará obrigatoriamente um objeto da classe `Setor`, a partir de uma execução parametrizada do método `select()` da implementação de `TableGatewayInterface`. Será gerada para execução uma expressão do tipo `SELECT * FROM [NOME DA TABELA] WHERE [$this->keyName] = [$keyValue]`. Se existir um registro cuja chave primária corresponda ao valor de `keyValue`, será retornado um objeto com os valores do registro encontrado. Se não, será retornado um objeto com atributos nulos.

Em seguida, adicione as linhas adiante:

```
    /**
     *
     * @param Setor $setor
     */
    public function saveModel(Setor $setor)
    {
        $data = array(
            'nome' => $setor->nome
```

```
        );
        $codigo = $setor->codigo;
        if (empty($this->getModel($codigo)->codigo)) {
            $data['codigo'] = $codigo;
            $this->tableGateway->insert($data);
        } else {
            $this->tableGateway->update($data, array(
                'codigo' => $codigo
            ));
        }
    }
```

O método `saveModel()` encapsula as operações de inclusão e atualização de registro, executadas respectivamente pelos métodos `insert()` e `update()` da implementação de `TableGatewayInterface`. O que define qual será executada é o valor do atributo `codigo` da instância de `Setor`.

Finalmente, acrescente as próximas linhas:

```
    /**
     *
     * @param mixed $keyValue
     */
    public function deleteModel($keyValue)
    {
        $this->tableGateway->delete(array(
            $this->keyName => $keyValue
        ));
    }
}
```

O método `deleteModel()` é uma fachada para o `delete()` da implementação de `TableGatewayInterface`, que remove um registro de uma tabela de acordo com a expressão de filtro passada como argumento. Neste caso, o array representa uma condição de igualdade, de modo a gerar uma expressão do tipo `DELETE FROM [NOME DA TABELA] WHERE [$this->keyName] = [$keyValue]`.

Vamos falar mais sobre a interface `TableGatewayInterface`. Ela define o comportamento para uma implementação do padrão de projeto *Table Data Gateway*.

Segundo Fowler (2006, p. 151), Table Data Gateway define “um objeto que atua como um Gateway para uma tabela do banco de dados”, uma instância que “lida com todas as linhas na tabela”, para efetuar as operações sobre a tabela mapeada. O uso desse padrão é necessário para permitir a manipulação de uma tabela de banco de dados (ou parte dela) como um objeto em memória. Ele é útil para manter a uniformidade em um sistema de software orientado a objetos.

A classe `Laminas\TableGateway` é uma implementação padrão do Laminas para `TableGatewayInterface`. Mas como a classe `SetorTable` sabe que deve manipular a tabela `setor` do banco `corps`? Ainda não sabe. Veremos isso a seguir.

Por enquanto, apenas criamos duas classes na camada de modelo que contêm uma representação de uma entidade do negócio (modelo) e as operações que podem ser realizadas sobre os dados dessa entidade (mapeador).

7.2 Conexão e mapeamento objeto-relacional como serviço

A conexão com o banco de dados e o efetivo mapeamento entre a tabela `setor` e a classe-modelo `Setor` será feita pela classe `ServiceManager` do Laminas. A `ServiceManager` pode ser configurada no arquivo `module.config.php` do módulo, neste caso, localizado em `module/Tropa/config`. Edite-o e acrescente a chave `service_manager` ao array, com o seguinte conteúdo:

module.config.php=>service_manager

```
    'service_manager' => [
            'factories' => [
                SetorTable::class =>  function($sm) {
                    $tableGateway = $sm->get('SetorTableGateway');
                    $table = new SetorTable($tableGateway);
                    return $table;
                },
                'SetorTableGateway' => function ($sm) {
                    $dbAdapter = $sm->get('Laminas\Db\Adapter');
                    $resultSetPrototype = new ResultSet();
                    $resultSetPrototype->setArrayObjectPrototype(new
Setor());
                    return new TableGateway('setor', $dbAdapter, null,
$resultSetPrototype);
                },
            ],
    ]
```

A variável `$sm` representa a instância de `ServiceManager`, que vai gerenciar o mapeamento objeto-relacional. O método `get()` invocado na closure (indicada pela chave `Tropa\Model\SetorTable`) vai recuperar outra closure, que retornará um gateway que utiliza o modelo `Setor` como objeto correspondente aos registros da tabela.

Essas dependências exigirão algumas importações, de modo que a seção de `uses` do arquivo fique assim:

```
use Laminas\Router\Http\Segment;
use Laminas\ServiceManager\Factory\InvokableFactory;
use Tropa\Model\SetorTable;
use Laminas\Db\ResultSet\ResultSet;
use Tropa\Model\Setor;
use Laminas\Db\TableGateway\TableGateway;
```

Preste uma atenção especial aos namespaces. Se você não declarar o namespace de uma classe e esta estiver dentro de um namespace, ela considerará todas as referências a outras classes como sendo de seu próprio namespace. Por exemplo, se não houver a declaração do namespace da classe `SetorTable` no

cabeçalho do arquivo da classe `Module`, será assumido o namespace `Tropa\SetorTable`, que não existe.

As classes `ResultSet` e `TableGateway` fazem parte do componente `Laminas\Db`, por isso precisamos incluí-lo em nosso projeto. Edite o arquivo `composer.json` e altere a chave `require`, de modo que fique assim:

composer.json=>require

```
"require" : {
        "php" : "^7.3",
        "laminas/laminas-component-installer" : "^1.0 || ^2.1",
        "laminas/laminas-development-mode" : "^3.2",
        "laminas/laminas-skeleton-installer" : "^0.1.7 || ^1.0",
        "laminas/laminas-mvc" : "^3.1.1",
        "laminas/laminas-mvc-i18n": "^1.1",
        "laminas/laminas-test": "^3.4",
        "laminas/laminas-db": "^2.11"
    },
```

Execute o comando `composer update`, e depois atualize o projeto com `F5` para que o Eclipse rastreie os arquivos do componente adicionado. Na execução do Composer, é possível selecionar a opção 1 para o `Laminas\Db`, para que o componente já seja adicionado ao arquivo `modules.config.php`. Se você esquecer de escolher a opção 1 na atualização, pode incluir manualmente o componente no arquivo, como fizemos anteriormente.

Os dados de conexão com o banco de dados ficarão na configuração global da aplicação, no arquivo `global.php`, em `corps1\config\autoload`. Altere o conteúdo do array retornado por esse arquivo, como a seguir:

global.php

```
<?php
return [
    'db' => [
        'driver' => 'Pdo',
```

```
        'dsn' => 'mysql:dbname=corps;hostname=localhost',
        'driver_options' => [
            PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''
        ],
    ],
    'service_manager' => [
        'factories' => [
            'Laminas\Db\Adapter'
            => 'Laminas\Db\Adapter\AdapterServiceFactory',
        ],
    ],
];
```

Nesse arquivo, configuramos os dados de conexão e isolamos a dependência da aplicação com o banco de dados. Também configuramos a classe que o `ServiceManager` usará para fabricar as conexões – em outras palavras, instanciar a classe adaptadora do banco de dados utilizado.

A chave `db` é requerida pelo método `Laminas\Db\Adapter\AdapterService::createService`. Este é invocado por `ServiceManager` quando o valor associado a um nome de fábrica é identificado como uma classe.

Na verdade, estão faltando dois dados para a chave `db`, o nome e a senha do usuário do banco de dados. Eles ficarão em outro arquivo, o `local.php`. Para que isso? Para complicar a configuração? Longe disso, o motivo disso é separar a configuração de produção da configuração de desenvolvimento.

Os arquivos `global.php` e `local.php` são mesclados em um único array, mas você pode configurar o sistema de controle de versão para ignorar o arquivo `local.php` quando submeter mudanças ao repositório central do projeto. Assim, cada desenvolvedor pode ter seu usuário e senha para o banco de dados sem precisar compartilhar essa informação.

Só que não existe um `local.php`, e sim um `local.php.dist`. Copie o último e cole-o removendo a extensão `.dist`. Altere o array

retornado pelo arquivo desta forma:

local.php

```
<?php
return [
        'db' => [
                'username' => 'root',
                'password' => '[SENHA]',
        ],
];
```

Não se esqueça de colocar a senha do usuário root de seu servidor MySQL no lugar de `[SENHA]` . Se não tiver uma senha configurada, basta abrir e fechar apóstrofos. E se usar outro usuário, basta colocar o nome como valor da chave `username` .

7.3 Implementando as ações do controlador

Agora que a camada de modelo está implementada, podemos retornar à camada de controle. Vamos implementar o código que tornará os quatro métodos do controlador `SetorController` funcionais.

Antes de preencher os métodos criados, adicionaremos um método que retorne o mapeador, para que os demais possam manipular os dados do banco. Mas para que esse método faça isso, precisamos injetar essa dependência no controlador. Faremos isso ao alterar a fábrica do controlador `SetorController` no arquivo `module.config.php` do módulo `Tropa` , conforme a próxima listagem:

module.config.php=>controllers

```
    'controllers' => [
        'aliases' => [
            'setor' => Controller\SetorController::class
        ],
        'factories' => [
```

```
            Controller\IndexController::class => InvokableFactory::class,
            Controller\SetorController::class => function($sm){
                $table = $sm->get(SetorTable::class);
                return new Controller\SetorController($table);
            }
        ],
    ],
```

Edite a classe `SetorController` e acrescente o atributo e método a seguir:

SetorController->__construct()

```
    private $table;

    public function __construct($table)
    {
        $this->table = $table;
    }
```

O atributo `SetorController::table` será utilizado no método `SetorController::indexAction`, assim:

SetorController->indexAction()

```
    public function indexAction()
    {
        return new ViewModel(
            ['models' => $this->table->fetchAll()]
        );
    }
```

A página associada a essa ação, `index.phtml`, está no diretório `view/tropa/setor`. Lembre-se de que os arquivos `phtml` sempre estarão dentro do diretório `view`. Dentro dele, no primeiro nível, há um diretório que representa o módulo. No segundo nível, há diretórios que representam os controladores.

Vamos criar o `setor\index.phtml` passo a passo. Inicie adicionando as seguintes linhas:

```
<?php
$title = 'Setores ';
$this->headTitle($title);
?>
<h1><?=$this->escapeHtml($title)?></h1>
```

Observe que utilizamos neste arquivo o método `headTitle()` para definir o texto que será exibido na barra de títulos e na aba atual do navegador. É o mesmo método usado para imprimir o texto, no arquivo `layout.phtml`, do módulo `Application`. Este é um exemplo de método *faca de dois gumes*: ele opera de formas distintas de acordo com os argumentos passados (e se o retorno é usado).

Em seguida, adicione as próximas linhas:

```
<p>
    <a href="<?=$this->url(
        'tropa',
         array(
             'action'=>'edit',
             'controller'=>'setor'
            )
        )?>">Novo setor</a>
</p>
```

O método `url()` gera uma URL a partir dos parâmetros de uma rota, que deve estar definida no arquivo `module.config.php`. Esse método torna a navegação das páginas da aplicação independente da máquina em que estiver instalada, pois as URLs serão geradas dinamicamente.

Em seguida, adicione as linhas a seguir:

```
<table class="table">
<tr>
    <th>Código</th>
    <th>Nome</th>
    <th> </th>
</tr>
<?php foreach ($this->models as $model):?>
```

```
<tr>
    <td><?=$model->codigo?></td>
    <td><?=$this->escapeHtml($model->nome)?></td>
    <td>
        <a href="<?=$this->url('tropa',
            array(
                'action'=>'edit',
                'controller'=>'setor',
                'key' => $model->codigo
                )
            )?>">Editar</a>
        <a href="<?=$this->url('tropa',
            array(
                'action'=>'delete',
                'controller'=>'setor',
                'key' => $model->codigo
                )
            )?>">Remover</a>
    </td>
</tr>
<?php endforeach ?>
</table>
```

O conteúdo da tabela HTML é gerado dinamicamente, pois ele depende dos registros da tabela do banco de dados, que são mutáveis. O atributo `models` contém a coleção de registros passada pelo controlador. A estrutura de repetição `foreach` cria uma linha de tabela HTML para cada registro contido em `models`.

O método `escapeHTML()` realiza um tratamento de conteúdo para neutralizar caracteres (ou cadeias de caracteres) que possam executar ações maliciosas na apresentação de uma página HTML. Não devemos confiar na entrada de dados nem na saída.

A tabela gerada é formada por duas colunas que contêm links para duas ações do controlador: alteração e exclusão.

Pronto, já estabelecemos a conexão da aplicação com um banco de dados relacional e o mapeamento das tabelas para classes. No próximo capítulo, aprenderemos a construir formulários dinâmicos,

cujo preenchimento está conectado aos modelos e aos mapeadores objeto-relacionais que os populam.

Capítulo 8
Formulários dinâmicos

O mais importante é a mudança, o movimento, o dinamismo, a energia. Só o que está morto não muda!. – Clarice Lispector

Neste capítulo, vamos conhecer o componente de geração de formulários dinâmicos, o `Laminas\Form`, que serve de conexão entre as camadas de visão e de modelo.

Formulários dinâmicos são formulários criados em tempo de execução. Isso quer dizer que eles não estão definidos previamente em arquivos HTML, mas, sim, que serão gerados pela aplicação no momento em que forem usados. O motivo de trabalhar com formulários dinâmicos em vez de estáticos é a flexibilidade de alterar a interface com o usuário de acordo com condições que podem se modificar, como o acréscimo de campos a uma tabela do banco de dados.

8.1 Criando formulários dinâmicos

A página que servirá para incluir e alterar os dados de setores precisará de um conjunto de campos que geralmente estão reunidos dentro de um formulário. Criaremos uma classe que definirá um formulário de inclusão para os setores usando o componente `Laminas\Form`.

Essa classe terá em seu construtor chamadas a métodos que definirão os elementos do formulário. Esses elementos estarão encapsulados em objetos e serão adicionados a um objeto que encapsulará o formulário completo. Sendo assim, em `module\Tropa\src\Form`, vamos criar a classe `Setor` passo a passo.

Crie o arquivo `Setor.php` com as seguintes linhas:

```php
<?php
namespace Tropa\Form;

use Laminas\Form\Form;
class Setor extends Form
{
    public function __construct($name = null) {
        parent::__construct('setor');
        $this->setAttribute('method', 'post');
```

O construtor de `Laminas\Form\Form` recebe como primeiro argumento o valor dos atributos `id` e `name` do formulário HTML. Qualquer outro atributo do formulário pode receber um valor por meio do método `setAttribute()`, no qual o primeiro argumento é o nome do atributo e o segundo, seu valor.

Apenas este trecho inicial do método, `__construct()` gerará o seguinte bloco de código HTML:

```html
<form method="post" name="setor" id="setor">
</form>
```

Vamos adicionar as próximas linhas, completando o conteúdo do método construtor:

```php
        $this->add([
            'name' => 'codigo',
            'attributes' => [
                'type'  => 'number',
                'autofocus'=>'autofocus'
            ],
            'options' => [
                'label' => 'Código',
            ]
        ]);
        $this->add([
            'name' => 'nome',
            'attributes' => [
                'type'  => 'text',
```

```
            ],
            'options' => [
                'label' => 'Nome',
            ],
        ]);
        $this->add([
            'name' => 'submit',
            'attributes' => [
                'type'  => 'submit',
                'value' => 'Gravar',
                'id' => 'submitbutton',
            ],
        ]);
    }
}
```

Os elementos do formulário são adicionados com o método `add()`, que, neste caso, recebe como argumento um array que define o elemento. Esse array deve obrigatoriamente definir duas chaves: `name` e `attributes`. A primeira indica o valor dos atributos `id` e `name` do elemento. A segunda indica o valor dos demais atributos, sendo que o mais importante é o `type`, que indica qual é o tipo de elemento.

Outros argumentos do elemento, como o rótulo que o antecederá, são definidos pela chave `options`. Neste caso, estamos criando três campos de entrada de dados para formulário HTML: um do tipo `number`, um do tipo `text` e um do tipo `submit`.

O código gerado pelas duas chamadas anteriores ao método `add()` será algo assim:

```
    <fieldset>
      <label>
        <span>Código</span>
        <input name="codigo" type="number" autofocus="autofocus" value="">
      </label>
      <label>
        <span>Nome</span>
        <input name="nome" type="text" value="">
```

```
      </label>
      <input name="submit" type="submit" id="submitbutton"
value="Cadastrar">
    </fieldset>
```

Após a definição da classe, o Eclipse vai indicar que não encontrou o componente `Laminas\Form`. Por isso, precisamos incluir a dependência no arquivo `composer.json`, conforme a listagem a seguir:

composer.json=>require

```
"require" : {
        "php" : "^7.3",
        "laminas/laminas-component-installer" : "^1.0 || ^2.1",
        "laminas/laminas-development-mode" : "^3.2",
        "laminas/laminas-skeleton-installer" : "^0.1.7 || ^1.0",
        "laminas/laminas-mvc" : "^3.1.1",
        "laminas/laminas-mvc-i18n": "^1.1",
        "laminas/laminas-test": "^3.4",
        "laminas/laminas-db": "^2.11",
        "laminas/laminas-form": "^2.15"
    },
```

Execute o comando `composer update`, escolhendo a opção 1 quando questionado. Em seguida, atualize o projeto no Eclipse, para que o IDE encontre as referências ao componente `Laminas\Form`.

A classe `Laminas\Form\Form` suporta elementos-padrão e elementos HTML5. Veremos mais sobre eles adiante.

Validando formulários

Esse formulário será validado. Validação é regra de negócio, portanto, responsabilidade do modelo. Então, nada mais lógico do que implementar a validação no modelo. Edite o arquivo `Setor.php` que está em `module\Tropa\src\Model`. Vamos alterá-lo passo a passo.

Primeiro, vamos modificar os namespaces usados. As primeiras linhas do arquivo, até o início da declaração da classe, devem ficar assim:

```php
<?php
namespace Tropa\Model;

use Laminas\InputFilter\InputFilterInterface;
use Laminas\InputFilter\InputFilter;
use Laminas\InputFilter\Factory;

class Setor
{
```

A interface `InputFilterInterface` define métodos para um objeto que encapsula ações de filtro e validação de dados para um conjunto de entradas. A classe `InputFilter` é a implementação padrão do ZF3 para a interface `InputFilterInterface`. A classe `Factory` do componente `Laminas\InputFilter` cria, a partir um array de argumentos, um objeto da classe `Input`, que corresponde a uma entrada de dados.

Após a abertura da declaração da classe, mantemos os atributos do modelo que correspondem aos campos na tabela do banco de dados, como se vê a seguir:

```php
    /**
     *
     * @var integer
     */
    public $codigo;

    /**
     *
     * @var string
     */
    public $nome;
```

Porém, vamos adicionar um atributo que encapsulará a implementação de `InputFilterInterface`. Então, após a declaração do atributo `nome`, acrescente as seguintes linhas:

```php
    /**
     *
```

```
     * @var InputFilterInterface
     */
    private $inputFilter;
```

O método `exchangeArray()`, que serve para popular os atributos da classe, se mantém conforme o trecho a seguir:

```
    /**
     *
     * @param array $data
     */
    public function exchangeArray(array $data)
    {
        foreach($data as $attribute => $value){
            $this->$attribute = $value;
        }
    }
```

Em seguida, adicionamos passo a passo um novo método, `getInputFilter()`, que retornará uma instância de `InputFilterInterface`, ou mais especificamente uma instância de `InputFilter`. Primeiro adicione as seguintes linhas:

```
    /**
     *
     * @return \Laminas\InputFilter\InputFilterInterface
     */
    public function getInputFilter()
    {
        if (! $this->inputFilter) {
```

O método começa com um `if` porque adotamos a estratégia de carregamento tardio: só vamos construir o objeto quando ele for usado. Se não houver necessidade de usar o atributo `inputFilter`, ele permanecerá nulo e não usaremos a memória inutilmente para guardá-lo.

Vamos prosseguir a construção do método `getInputFilter()` com as seguintes linhas:

```
$inputFilter = new InputFilter();
$factory = new Factory();
```

A variável `inputFilter` contém um objeto da classe `InputFilter`, inicialmente vazio, sem filtros e sem regras de validação definidas. A variável `factory` contém um objeto que sabe como criar um objeto de entrada de dados, que encapsula filtros e regras de validação para um campo de entrada.

Continue a construção do método adicionando as linhas seguintes:

```
$inputFilter->add($factory->createInput([
    'name' => 'codigo',
    'required' => false,
    'filters' => [
        [
            'name' => 'Int'
        ]
    ],
    'validators' => [
        [
            'name' => 'Between',
            'options' => [
                'min' => 0,
                'max' => 3600
            ]
        ]
    ]
]));
```

O método `add()` de `InputFilter` acrescenta um campo de entrada de dados para o gerenciador de filtros e validadores. O método `createInput()` de `Factory` retorna um objeto da classe `Input` a partir do array passado como argumento. Um objeto da classe `Input` pode conter um conjunto de filtros, que fazem transformações em dados, e um conjunto de regras de validação, que dizem se os dados recebidos pelo objeto são válidos.

Neste caso, o campo `codigo` terá seu valor forçado para inteiro e só permitirá números com quatro dígitos no intervalo entre 0 e 3.600,

incluindo os dois extremos.

A seguir, adicione a definição de filtros e validadores para o segundo campo manipulado pelo modelo `Setor`, o campo `nome`:

```
            $inputFilter->add($factory->createInput([
                'name' => 'nome',
                'required' => true,
                'filters' => [
                    [
                        'name' => 'StripTags'
                    ],
                    [
                        'name' => 'StringTrim'
                    ]
                ],
                'validators' => [
                    [
                        'name' => 'StringLength',
                        'options' => [
                            'encoding' => 'UTF-8',
                            'min' => 2,
                            'max' => 30
                        ]
                    ]
                ]
            ]));
```

O campo `nome` terá extirpado de seu conteúdo caracteres delimitadores de tag e espaços no início e no fim do texto, e só permitirá palavras com até 30 caracteres.

Finalmente, encerramos o método fechando a estrutura `if` e retornando o objeto, tenha ele sido criado naquele momento ou anteriormente:

```
                $this->inputFilter = $inputFilter;
            }
            return $this->inputFilter;
        }
}
```

Acrescentamos um filtro de entrada com conversões e algumas regras de validação para os dois únicos campos da tabela. Caso você tenha piscado, isso está implementado no método `getInputFilter()`.

Formulário de cadastro

Agora precisamos utilizar esse formulário. Vamos preencher o método `editAction()` de `SetorController` com o seguinte código-fonte:

SetorController->editAction()

```
    /**
     * Action to add/edit and change records
     */
    public function editAction()
    {
        $form = new SetorForm();
        $form->get('submit')->setValue('Cadastrar');
        return ['form' => $form];
    }
```

No método `editAction()`, instanciamos a classe `SetorForm` e configuramos o elemento do tipo `submit` para o valor "Cadastrar". A classe `SetorForm` não existe de verdade, é um apelido para `Tropa\Form\Setor`. Por isso, para que a referência funcione, é necessária a associação de um apelido com o namespace da classe no cabeçalho do arquivo. Os namespaces usados pelo controlador neste momento serão os seguintes:

```
use Laminas\Mvc\Controller\AbstractActionController;
use Laminas\View\Model\ViewModel;
use Tropa\Form\Setor as SetorForm;
```

Se você estiver usando o autocompletamento do Eclipse para digitar o nome das classes, as importações de namespaces serão feitas automaticamente se você digitar `CTRL + BARRA DE ESPAÇO` após o nome da classe.

Para deixar ainda mais claro, quando um método de controlador terminado com o sufixo `Action` for encerrado, caso não ocorra um redirecionamento para outro método, o processamento passará para a camada de visão, e um arquivo `.phtml` – cujo nome corresponde ao método do controlador – será fundido ao layout e executado.

Neste caso, o arquivo `edit.phtml`, criado neste capítulo, deve ser preenchido com o código-fonte a seguir:

setor\edit.phtml

```
<?php
$title = 'Incluir o setor';
$this->headTitle($title);
?>
<h1><?=$this->escapeHtml($title)?></h1>
<?php
$form = $this->form;
$form->setAttribute('action', $this->url(
                    'tropa',
                     [
                         'action' => 'save',
                         'controller' => 'setor'
                    ]));
$form->prepare();

echo $this->form()->openTag($form);
echo $this->formRow($form->get('codigo'));
echo $this->formRow($form->get('nome'));
echo $this->formSubmit($form->get('submit'));
echo $this->form()->closeTag();
```

O método `setAttribute()` configura o valor de um atributo para a tag `FORM` de HTML. O método `prepare()` garante que as mensagens de erro de validação dos elementos do formulário estarão disponíveis para exibição. Já o `openTag()` gera a tag de abertura de formulário (`<form>`).

O método `formRow()` gera uma tag de entrada de dados (que seja digitável) a partir de um objeto da classe `Element` do componente `Laminas\Form` . O `formSubmit()` gera uma tag `INPUT` com `type` igual a `submit` , a partir de um objeto da classe `Submit` do componente `Laminas\Form` . Por fim, o método `closeTag()` gera a tag de fechamento de formulário (`</form>`).

Após implementar a classe `Tropa\Form\Setor` , você talvez esperasse uma implementação mais simples. A impressão é que estamos tendo trabalho repetido. Já não adicionamos os elementos? A ordem deles já não será a mesma em que foram adicionados? Os atributos do formulário não foram definidos? Para que escrever dessa forma?

A resposta é mostrar como tornar a implementação mais flexível. Essa forma de construir a visualização do formulário permite alterar a ordem dos elementos e, inclusive, acrescentar elementos que não foram definidos na extensão `Laminas\Form\Form` .

Se a flexibilidade não é preocupação na construção dos formulários, há uma forma mais concisa. Agora que conhecemos a forma mais prolixa, vamos adotar a concisa, editando o arquivo e substituindo seu conteúdo pelo seguinte código-fonte:

setor\edit.phtml

```
<?php
$title = 'Incluir o setor';
$this->headTitle($title);
?>
<h1><?=$this->escapeHtml($title)?></h1>
<?php
```

```
$form = $this->form;
$form->setAttribute('action', $this->url(
                    'tropa',
                     [
                         'action' => 'save',
                         'controller' => 'setor'
                    ]));
$form->prepare();

echo $this->form($form);
```

Com isso, o método `form()` substituiu todos os `echo` anteriores referentes aos elementos do formulário.

Vamos adicionar um estilo para os elementos de formulário que usaremos em nosso projeto. No arquivo `tropa.css` criado anteriormente, acrescente o seguinte estilo:

```
input,select {
    display: block;
}
```

Isso é para que os campos de entrada não sejam renderizados em uma única linha. Bem, se um setor tiver de ter seu código ou nome alterados, não vamos apagá-lo e incluí-lo novamente, certo?

O método `editAction()` servirá para incluir e alterar. No momento, ele só serve para incluir. Para alterar, teremos de modificá-lo de modo que ele fique assim:

SetorController->editAction()

```
    public function editAction()
    {
        $codigo = $this->params()->fromRoute('key', null);
        $setor = $this->table->getModel($codigo);
        $form = new SetorForm();
        $form->get('submit')->setValue(
            empty($codigo) ? 'Cadastrar' : 'Alterar'
            );
        $form->bind($setor);
```

```
        return ['form' => $form];
    }
```

O método `fromRoute()` retorna o valor de um parâmetro em uma URL de acordo com a configuração da rota. Se o parâmetro não estiver presente na URL, será retornado `NULL` por padrão.

O método `getModel()` nós mesmos criamos, junto com a classe `SetorTable`. Ele retorna um objeto da classe `Setor`.

Como o mesmo formulário é usado para incluir e alterar, temos de verificar com a função `empty()` do PHP se o código do setor é nulo para saber qual será o rótulo do botão de submissão.

O método `bind()` transfere os valores dos atributos de um objeto modelo para os campos correspondentes em um objeto formulário. Para que o método `bind()` de `Laminas\Form\Form` funcione neste caso, precisamos implementar o seguinte método na classe-modelo `Setor`:

Setor->getArrayCopy()

```
    /**
     *
     * @return array
     */
    public function getArrayCopy() {
        return [
            'codigo'=>$this->codigo,
            'nome'=>$this->nome
        ];
    }
```

O método `bind()` faz uma chamada a um método `getArrayCopy()` para obter os valores dos campos do formulário. O método `getArrayCopy()` deve retornar um array com os atributos do objeto.

O método `editAction()` exibe o formulário para incluir ou alterar setores. Esse formulário submete os dados para o método `saveAction()`. Isso foi definido no arquivo `edit.phtml`, por meio do

método `setAttribute()` da classe `Laminas\Form`. Vamos implementá-lo a seguir.

SetorController->saveAction()

```
/**
 * Action to save a record
 */
public function saveAction()
{
    $request = $this->getRequest();
    if ($request->isPost()) {
        $form = new SetorForm();
        $setor = new Setor();
        $form->setInputFilter($setor->getInputFilter());
        $form->setData($request->getPost());
        if ($form->isValid()) {
            $setor->exchangeArray($form->getData());
            $this->table->saveModel($setor);
        }
    }
    return $this->redirect()->toRoute(
        'tropa',
        ['controller'=>'setor']
        );
}
```

Usamos o método `getRequest()` para obter um objeto `Laminas\HttpRequest`. Esse objeto, entre outras coisas, informa qual é o tipo de requisição HTTP. Assim, verificamos se uma requisição HTTP `POST` foi enviada com o método `isPost()`.

Em caso positivo, um modelo `Setor` é instanciado, e o objeto formulário é alimentado com os filtros e as regras de validação (provenientes de `SetorForm`), com o método `setInputFilter()`, e dos dados oriundos da requisição HTTP, com o método `setData()`. Ao invocarmos `isValid()`, o objeto `SetorForm` responde se os dados são válidos, e assim podemos usá-los para preencher os atributos do objeto `Setor`. Nesse caso, ocorre um redirecionamento para a rota

`setor` com o método `redirect()`, que invocará o método `indexAction()` de `SetorController`.

Caso a requisição não seja HTTP `POST` ou os dados sejam inválidos, o retorno do método será um array contendo o objeto `SetorForm`. Saberemos o motivo disso mais adiante. Mas espere, não deveria retornar um objeto `ViewModel`? Na verdade, o array é encapsulado em um `ViewModel`, de forma implícita. Se você não vai configurar nada além de variáveis a serem exibidas na página, pode usar apenas um array.

Finalmente, vamos implementar a mais fácil de todas as operações, a que remove o setor da tabela. O método `deleteAction()` de `SetorController` ficará assim:

SetorController->deleteAction()

```
/**
 * Action to remove records
 */
public function deleteAction()
{
    $codigo = $this->params()->fromRoute('key', null);
    $this->table->deleteModel($codigo);
    return $this->redirect()->toRoute(
        'tropa',
        ['controller'=>'setor']
        );
}
```

Esse método é acionado quando o usuário clica sobre o link de exclusão de um setor na página construída pelo arquivo `index.phtml`, que exibe a lista de setores cadastrados. Ele usa o método `deleteModel()` (da classe `SetorTable`) para apagar o registro que tem como a chave primária igual ao valor da variável `codigo`.

Finalmente, ele usa o método `redirect()` para redirecionar o usuário para a página de listagem dos setores. Esse método recupera o valor do código contido na URL do link com o método `fromRoute()`.

Não esqueça de verificar se o namespace do modelo está incluído na lista de importações do controlador:

```
use Tropa\Model\Setor;
```

Com a implementação das quatro operações que compõem o nosso cadastro de setores, podemos testar as funcionalidades de nosso módulo, incluindo alguns setores. Neste momento, a URL http://localhost:8000/tropa/setor deve apresentar a página da figura a seguir.

Figura 8.1: Saída da página index.phtml de SetorController

Clique no link `Novo Setor` , e será exibida a página da próxima figura, com o formulário para inclusão de setores espaciais.

Figura 8.2: Saída da página edit.phtml de SetorController

8.2 Validando formulários dinâmicos

A regra de negócio referente ao código é a seguinte: existem 3.601 setores espaciais (do 0 ao 3.600). O campo código deve ser preenchido, porque o código é a chave primária da tabela de setores. O nome, por sua vez, está restrito a um texto entre dois e trinta caracteres. Por isso, o primeiro teste é tentar inserir dados que firam essas regras e ver se serviu para alguma coisa criar as regras de validação no modelo e depois incorporá-las no formulário.

Mas para que as mensagens de validação sejam exibidas, precisamos capturá-las no momento da submissão do formulário e recuperá-las na reexibição do formulário. Para isso, teremos de modificar os métodos `editAction()` e `saveAction()`, de acordo com as listagens a seguir.

Na alteração do método `editAction`, precisaremos da classe `Laminas\Session\Container`, que permite o armazenamento de objetos

na sessão. Para usá-la, precisamos baixar o componente `Laminas\Session`. Altere a seção `require` do arquivo `composer.json`, para que reflita a seguinte listagem:

composer.json=>require

```
"require" : {
        "php" : "^7.3",
        "laminas/laminas-component-installer" : "^1.0 || ^2.1",
        "laminas/laminas-development-mode" : "^3.2",
        "laminas/laminas-skeleton-installer" : "^0.1.7 || ^1.0",
        "laminas/laminas-mvc" : "^3.1.1",
        "laminas/laminas-mvc-i18n": "^1.1",
        "laminas/laminas-test": "^3.4",
        "laminas/laminas-db": "^2.11",
        "laminas/laminas-form": "^2.15",
        "laminas/laminas-session": "^2.9"
    },
```

Em seguida, execute o comando `composer update`, respondendo com a opção 1 quando questionado sobre a injeção do componente. Não esqueça de atualizar o projeto no Eclipse, para que ele recupere as referências.

Agora, altere o método `editAction()`, de modo que ele corresponda à listagem a seguir:

SetorController->editAction()

```
    public function editAction()
    {
        $codigo = $this->params()->fromRoute('key', null);
        $setor = $this->table->getModel($codigo);
        $form = new SetorForm();
        $form->get('submit')->setValue(
            empty($codigo) ? 'Cadastrar' : 'Alterar'
            );
        $sessionContainer = new SessionContainer();
        if (isset($sessionContainer->model)){
            $setor->exchangeArray($sessionContainer->model->toArray());
            unset($sessionContainer->model);
```

```
            $form->setInputFilter($setor->getInputFilter());
        }
        $form->bind($setor);
        $form->isValid();
        return ['form' => $form];
    }
```

Essa alteração exige uma importação da classe `Container` de `Laminas\Session` com o apelido de `SessionContainer` no cabeçalho da classe `SetorController`.

```
use Laminas\Session\Container as SessionContainer;
```

Nessa alteração, verificamos se o objeto `sessionContainer` tem um atributo `model` com a função `isset()` do PHP. Caso tenha, quer dizer que há um modelo armazenado na sessão e preenchemos o formulário com os valores desse modelo.

Quem armazena o modelo na sessão é o método `saveAction()`, que alteramos a seguir:

SetorController->saveAction()

```
    public function saveAction()
    {
        $request = $this->getRequest();
        if ($request->isPost()) {
            $form = new SetorForm();
            $setor = new Setor();
            $form->setInputFilter($setor->getInputFilter());
            $post = $request->getPost();
            $form->setData($post);
            if (!$form->isValid()) {
                $sessionContainer = new SessionContainer();
                $sessionContainer->model = $post;
                return $this->redirect()->toRoute(
                    'tropa',
                    [
                        'action'=>'edit',
                        'controller'=>'setor'
                    ]
```

```
                    );

            }
            $setor->exchangeArray($form->getData());
            $this->table->saveModel($setor);
        }
        return $this->redirect()->toRoute(
            'tropa',
            ['controller'=>'setor']
            );
    }
```

Usamos o método `isValid()` para verificar se os dados do modelo são válidos. Caso não sejam, o objeto é armazenado na sessão por `SessionContainer`, e o usuário é redirecionado para o método `editAction()`.

A classe `SessionContainer` oferece uma interface orientada a objetos para armazenamento de dados na sessão. Para que ela funcione, precisamos que a sessão tenha sido iniciada. Vamos iniciar a sessão no construtor da classe `SetorController`, usando injeção de dependência.

Primeiro, alteraremos a fábrica da classe `SetorController` no arquivo `module.config.php`. Edite a seção `controllers` para que reflita a listagem a seguir:

module.config.php=>controllers

```
    'controllers' => [
        'aliases' => [
            'setor' => Controller\SetorController::class
        ],
        'factories' => [
            Controller\IndexController::class => InvokableFactory::class,
            Controller\SetorController::class => function($sm){
                $table = $sm->get(SetorTable::class);
                $sessionManager = new SessionManager();
                return new Controller\SetorController($table,
$sessionManager);
```

```
            }
        ],
    ],
```

Não esqueça de conferir se o namespace da classe `SessionManager` está devidamente citado no cabeçalho do arquivo `module.config.php`:

```
use Laminas\Session\SessionManager;
```

O próximo passo é alterar o construtor da classe `SetorController` para que seja compatível com a alteração feita na fábrica:

SetorController->__construct()

```
    public function __construct($table, $sessionManager)
    {
        $this->table = $table;
        $sessionManager->start();
    }
```

Agora, se tentarmos incluir um setor 3.601 com nome em branco, teremos como resposta a página da figura a seguir.

localhost:8000/tropa/setor/edit
Sistema de Controle da Tropa dos Lanternas Verdes
Incluir o setor
Código
3601
• The input is not between '0' and '3600', inclusively
Nome
• Value is required and can't be empty
Cadastrar
© 2020 by Oa Sector 0 All rights reserved.

Figura 8.3: Página edit.phtml com mensagens de validação

Observe que o formulário foi reapresentado, mantendo os dados que foram digitados, e exibiu mensagens de acordo com as regras não cumpridas. Isso ocorre porque o método `editAction()` retorna o objeto `SetorForm` após ter invocado o `isValid()`. A invocação desse método altera a apresentação do formulário, caso alguma regra de validação associada tenha sido descumprida.

Mensagens de validação em português

Legal, temos mensagens de validação, sem muito trabalho. Só que elas estão em inglês. Como fazemos para apresentá-las em português? Muito simples.

Vamos injetar um objeto de tradução na classe abstrata dos validadores com as traduções. Certo, mas como fazemos isso? Ou melhor, onde fazemos isso?

Primeiro, baixaremos o componente que contém as traduções das mensagens de validação. Abra o arquivo `composer.json` e altere a seção `require` para baixar o componente `Laminas\I18n\Resources`:

composer.json=>require

```
"require" : {
    "php" : "^7.3",
    "laminas/laminas-component-installer" : "^1.0 || ^2.1",
    "laminas/laminas-development-mode" : "^3.2",
    "laminas/laminas-skeleton-installer" : "^0.1.7 || ^1.0",
    "laminas/laminas-mvc" : "^3.1.1",
    "laminas/laminas-mvc-i18n": "^1.1",
    "laminas/laminas-test": "^3.4",
    "laminas/laminas-db": "^2.11",
    "laminas/laminas-form": "^2.15",
    "laminas/laminas-session": "^2.9",
    "laminas/laminas-i18n-resources": "^2.6"
},
```

Execute o comando `composer update` e não se esqueça de atualizar o projeto no Eclipse. Precisamos da tradução no momento de

reapresentar o formulário, o que é feito em `SetorController::editAction`. Esse método chamará outro método, protegido, que fará a injeção do tradutor na validadora-mãe, conforme a próxima listagem:

SetorController->initValidatorTranslator()

```
    protected function initValidatorTranslator()
    {
        $translator = new Translator();
        $mvcTranslator = new MvcTranslator($translator);
        $mvcTranslator->addTranslationFilePattern(
            'phparray',
            Resources::getBasePath(),
            Resources::getPatternForValidator()
        );

        AbstractValidator::setDefaultTranslator($mvcTranslator);
    }
```

Observe que esse método exigirá alguns namespaces no cabeçalho do arquivo:

```
use Laminas\Mvc\I18n\Translator as MvcTranslator;
use Laminas\Validator\AbstractValidator;
use Laminas\I18n\Translator\Translator;
use Laminas\I18n\Translator\Resources;
```

Em seguida, altere o método `SetorController::editAction`, para que chame `SetorController::initValidatorTranslator` dessa forma:

SetorController->editAction()

```
    public function editAction()
    {
        $codigo = $this->params()->fromRoute('key', null);
        $setor = $this->table->getModel($codigo);
        $form = new SetorForm();
        $form->get('submit')->setValue(
            empty($codigo) ? 'Cadastrar' : 'Alterar'
            );
```

```
        $sessionContainer = new SessionContainer();
        if (isset($sessionContainer->model)){
            $setor->exchangeArray($sessionContainer->model->toArray());
            unset($sessionContainer->model);
            $form->setInputFilter($setor->getInputFilter());
            $this->initValidatorTranslator();
        }
        $form->bind($setor);
        $form->isValid();
        return ['form' => $form];
    }
```

Após essa alteração, se você tentar novamente cadastrar o setor 3601 com nome vazio, verá a página da figura a seguir.

localhost:8000/tropa/setor/edit
Sistema de Controle da Tropa dos Lanternas Verdes
Incluir o setor
Código
3601
• O valor de entrada não está entre '0' e '3600', inclusivamente
Nome
• O valor é obrigatório e não pode estar vazio
Cadastrar
© 2020 by Oa Sector 0 All rights reserved.

Figura 8.4: Página edit.phtml com mensagem traduzida

O idioma é selecionado da localização, que é obtida automaticamente a partir da configuração do navegador; foi por isso que não configuramos explicitamente o idioma para português. Você pode forçar a localização usando as funções da API relacionada do PHP (funções com prefixo `locale`).

O que aconteceria se, mesmo indicando o arquivo de tradução e configurando a localidade para o idioma desejado, as mensagens

continuassem aparecendo no original em inglês? É bom saber sobre isso, pois eventualmente você pode usar um validador com problema na tradução.

A correção imediata é procurar no arquivo de tradução `Laminas_Validate.php`, na pasta do idioma que se deseja, pela mensagem original. Se ela não existir, então alguém esqueceu de traduzir, ou colocou a mensagem original com alguma palavra diferente, o que torna toda a mensagem diferente por consequência. As traduções ficam armazenadas na pasta `laminas-i18n-resources\languages` dentro de `vendor\laminas`.

Se você localizar a mensagem defeituosa, bastará corrigir o texto. Se não encontrar, terá de acrescentar a tradução. Lembre-se de que precisará submeter a correção para a comunidade do projeto `Laminas\I18n\Resources` se quiser atualizar o arquivo no futuro.

O arquivo de tradução das mensagens de validação é um array PHP, cujas chaves são as mensagens originais e os valores são as mensagens traduzidas. Se o validador aceita valores em seu construtor, eles podem ser exibidos na mensagem usando o delimitador `%`.

Por exemplo, o validador `StringLength` admite um argumento `min` em seu construtor, passado como uma chave de um array. Você pode conferir que esse valor é exibido na mensagem de validação no lugar da expressão `%min%`. A seguir, apresentamos a tradução da mensagem desse validador como um exemplo.

Exemplo de tradução de uma mensagem de validação

```
    // Laminas\Validator\StringLength
    "Invalid type given. String expected" => "O tipo especificado é inválido, o valor deve ser uma string",
    "The input is less than %min% characters long" => "O tamanho do valor de entrada é inferior a %min% caracteres",
    "The input is more than %max% characters long" => "O tamanho do valor de entrada é superior a %max% caracteres",
```

Agora que você sabe como traduzir as mensagens, pode incluir vários setores válidos, como 911 (Profundezas Obsidianas), 1417 (Korugar), 2261 (Mogo), 2814 (Terra) e 3182 (Daxam). Isso deixaria a URL http://localhost:8000/tropa/setor com uma aparência similar à da próxima figura.

localhost:8000/tropa/setor
Sistema de Controle da Tropa dos Lanternas Verdes
Setores
Novo setor
Código Nome
911 Profundezas Obsidianas Editar Remover
1417 Korugar Editar Remover
2261 Mogo Editar Remover
2814 Terra Editar Remover
3182 Daxam Editar Remover
© 2020 by Oa Sector 0 All rights reserved.

Figura 8.5: Página index.phtml com listagem de setores

8.3 Ajustes na alteração de setores

A listagem exibe cada registro com hyperlinks para alterar e excluir, mas eles ainda não funcionam adequadamente. Embora o método que gera a URL passe o código do setor como argumento, conforme fizemos anteriormente, a rota não foi preparada para recebê-lo e interpretá-lo. Por isso, vamos editar a rota default, filha de tropa, no arquivo `module.config.php`, para que admita um argumento chamado `key`.

module.config.php=>router

```
    'router' => [
        'routes' => [
            'tropa' => [
                'type' => Segment::class,
                'options' => [
                    'route'    => '/tropa[/:controller[/:action[/:key]]]',
                    'defaults' => [
                        'controller' => Controller\IndexController::class,
                        'action'     => 'index',
                    ],
                ],
            ],
        ],
    ],
```

Agora, após cautelosamente recarregar a página, você pode experimentar alterar e excluir setores e verá que implementamos um módulo seguindo o padrão MVC com grande sucesso.

Ainda existem alguns ajustes a fazer. Perceba que, ao abrir o formulário para alterar um setor existente, o título apresentado é 'Incluir o setor', como na adição de um novo setor. Isso ocorre porque a configuração do título está fixa no arquivo `edit.phtml` .

Vamos fazer com que o método `SetorController::editAction` determine o título e repasse-o ao arquivo `phtml` . Modifique o retorno do método `SetorController::editAction` para o seguinte:

setor\edit.phtml

```
        return [
            'form' => $form,
            'title' => empty($codigo) ? 'Incluir' : 'Alterar'
        ];
```

Esse retorno transmitirá a parte variável do título da página de edição. Para que o resultado manifeste-se, é necessário modificar as instruções do arquivo `edit.phtml` do controlador `SetorController` que manipulam o título para o seguinte conteúdo:

setor\edit.phtml

```
<?php
$title = $this->title . ' o setor';
$this->headTitle($title);
?>
<h1><?=$this->escapeHtml($title)?></h1>
```

Assim, você terá títulos adequados para cada situação. Lembre-se do óbvio: o controlador controla. E controlar é um verbo transitivo, pois alguém controla alguma coisa. No caso da classe controladora com a qual estamos lidando, ela implementa o padrão de projeto Controlador de Página (FOWLER, 2006), o que a responsabiliza diretamente pelo que as páginas exibirão.

Em outras palavras, um controlador de página é uma classe responsável pela exibição de um conjunto de páginas. Ela recebe requisições e, como comportamento padrão, apresenta uma página como resultado.

8.4 Ajustes na remoção de setores

Falta um último ajuste para terminar a primeira entrega de nosso projeto. Está muito fácil apagar setores, então, vamos adicionar uma confirmação para essa operação, para evitar enganos. Como a nossa remoção não é lógica (solução em que há uma coluna que indica se o registro foi excluído) e sim física, um clique errado resulta em um dado irreversivelmente perdido.

Primeiro, vamos alterar o tipo de elemento HTML usado para remover os setores. Em vez de um hyperlink renderizado pela tag `A`, vamos usar um botão renderizado pela tag `BUTTON`. Altere o trecho do arquivo `index.phtml` de `SetorController` que preenche a coluna de remoção de registros para este código:

setor\index.phtml

```
<button class="delete" url="<?=$this->url('tropa',
            array(
                'action'=>'delete',
                'controller'=>'setor',
                'key' => $model->codigo
                )
            )?>">Remover</button>
```

Isso fará com que a página se pareça com a próxima figura:

localhost:8000/tropa/setor
Sistema de Controle da Tropa dos Lanternas Verdes
Setores
Novo setor
Código Nome
911 Profundezas Obsidianas Editar Remover
1417 Korugar Editar Remover
2261 Mogo Editar Remover
2814 Terra Editar Remover
3182 Daxam Editar Remover
© 2020 by Oa Sector 0 All rights reserved.

Figura 8.6: Listagem de setores com botões para remoção

A tag `BUTTON` não tem o atributo `url` definido na especificação do HTML do W3C. Mas isso não importa. HTML é um tipo de XML, e você pode criar os atributos que quiser, não precisando se limitar à especificação. Esse atributo `url` serve para armazenar a URL para o qual vamos redirecionar se o usuário confirmar a remoção do registro.

Usaremos a biblioteca jQuery, importada pelo arquivo `layout.phtml`, para conectar o evento de clique no botão de remoção com uma função que peça a confirmação para o usuário. Na pasta `js` dentro de `public`, crie o seguinte arquivo `JavaScript`, o `public/delete.js`:

public\delete.js

```
/**
 * Show a confirmation window
 */
$(function() {
    $("button.delete").click(function() {
        if (window.confirm('Deseja realmente apagar?')) {
            window.location.replace($(this).attr("url"));
        }
    })
});
```

O `$` é o objeto seletor do jQuery. Ao passarmos uma função anônima no lugar de um seletor de elemento HTML, registramos uma ação a ser executada somente depois que todo o documento for carregado. Isso é necessário, pois não podemos associar uma ação a um evento se o elemento que dispara o evento ainda não existe.

Usamos um seletor para os elementos `BUTTON` com atributo `class` igual a `DELETE`, que conecta o evento de clique no botão a uma função anônima. Esta faz um redirecionamento para o endereço contido no atributo `url` do botão, caso o usuário confirme a operação.

Precisamos também que o arquivo `delete.js` seja importado pelo arquivo de layout. Faremos isso acrescentando uma instrução no arquivo `index.phtml` de `SetorController`. Será a primeira instrução do arquivo, no primeiro bloco PHP do arquivo, como no trecho a seguir:

setor\index.phtml

```
<?php
$this->headScript()->prependFile($this->basePath() . '/js/delete.js',
'text/javascript');
$title = 'Setores ';
$this->headTitle($title);
?>
```

O método `prependFile()` da classe auxiliar `Laminas\View\Helper\HeadScript` adiciona uma tag `SCRIPT` à tag `HEAD` do documento HTML. Isso quer dizer que qualquer visão pode acrescentar arquivos JavaScript ao documento, conforme sua necessidade.

O arquivo de layout não precisa importar todos os arquivos JavaScript, apenas os que sempre serão usados. Os arquivos de visões responsabilizam-se pelos comportamentos específicos de suas respectivas páginas.

Agora, ao clicar sobre o botão `Remover`, surgirá uma janela de confirmação:

Figura 8.7: Janela de confirmação de remoção

Acabamos nossa entrega? Não! Lembra de que planejamos dois cadastros, o de setores espaciais e o de Lanternas Verdes? Fizemos um deles, falta o outro. Além disso, nossa aplicação precisa de um menu inicial, por isso vamos alterar o módulo `Application`.

8.5 Modelo e mapeador com relacionamento

Vamos acrescentar uma tabela ao banco de dados `corps` para cadastrar os Lanternas Verdes. Se você tiver alguma dúvida sobre como fazer isso, consulte as seções anteriores deste capítulo. Utilize a seguinte declaração SQL para criar a tabela:

```
CREATE TABLE `lanterna` (
    `codigo` int(11) NOT NULL AUTO_INCREMENT,
    `nome` varchar(30) NOT NULL,
    `codigo_setor` int(11) NOT NULL,
    PRIMARY KEY (`codigo`)
);
```

Criaremos em seguida o modelo `Lanterna`. Este conterá métodos para entrada (`exchangeArray`) e saída de dados (`getArrayCopy`) e as regras de negócio referentes ao filtro e à validação dos dados (`getInputFilter`). Na pasta `module\Tropa\src\Model` , vamos criar o arquivo `Lanterna.php` . Construiremos seu conteúdo passo a passo.

Primeiro, adicione os namespaces e a abertura da declaração da classe:

```
<?php
namespace Tropa\Model;

use Laminas\InputFilter\Factory as InputFactory;
use Laminas\InputFilter\InputFilter;
use Laminas\InputFilter\InputFilterInterface;

class Lanterna
{
```

A classe `Factory` do componente `Laminas\InputFilter` já foi usada na classe modelo `Setor` com o mesmo propósito: criar objetos da classe `Input` . Aqui damos um apelido à classe (`InputFactory`) para que fique mais claro no código o que ela está fabricando.

A classe `InputFilter` também já foi utilizada na classe `Setor`. Ela é o gerenciador de filtros e validadores de uma coleção de entradas de dados.

A interface `InputFilterInterface` é implementada pela classe `Laminas\InputFilter\InputFilter`. Seu uso aqui é um ensaio de uma boa prática, a de preferir interfaces no lugar de implementações. Em nossas classes de modelo, estamos utilizando as implementações do Zend Framework, mas ao usar as interfaces na entrada e saída de dados dos métodos, facilitamos a possibilidade de uso de uma outra implementação no futuro.

Em seguida, criamos os atributos da classe:

```
    /**
     *
     * @var integer
     */
    public $codigo;

    /**
     *
     * @var string
     */
    public $nome;

    /**
     *
     * @var Setor
     */
    public $setor;

    /**
     *
     * @var InputFilterInterface
     */
    private $inputFilter;
```

Os atributos `codigo` e `nome` referem-se diretamente a campos encontrados na tabela `lanterna`. O atributo `setor`, entretanto, não

tem uma ligação direta com um campo da tabela. Ele trata-se de um mapeamento de uma chave estrangeira para um objeto. Enquanto a tabela `lanterna` relaciona-se com a tabela `setor` – por meio de um campo que contém valores em comum –, a classe `Lanterna` é composta por um objeto da classe `Setor`. A tabela `lanterna` não tem todos os dados do setor, enquanto a classe `Lanterna` tem.

O atributo `inputFilter`, à semelhança da classe `Setor`, não se refere a um campo da tabela mapeada, mas, sim, ao conjunto de filtros e regras de validação dos campos do modelo.

Em seguida, vamos criar o construtor da classe:

```
    public function __construct()
    {
        $this->setor = new Setor();
    }
```

No construtor, garantimos que, quando o objeto `Lanterna` for instanciado, ele conterá um objeto `Setor` vazio.

Em seguida, criamos o método para popular o modelo:

```
    /**
     *
     * @param array $data
     */
    public function exchangeArray($data)
    {
        $this->codigo = (isset($data['codigo'])) ? $data['codigo'] : null;
        $this->nome = (isset($data['nome'])) ? $data['nome'] : null;
        $this->setor = new Setor();
        $this->setor->codigo = (isset($data['codigo_setor'])) ?
$data['codigo_setor'] : null;
        $this->setor->nome = (isset($data['setor'])) ? $data['setor'] :
null;
    }
```

O método `exchangeArray()` recebe um array de dados, que devem ser variáveis de tipos primitivos do PHP. Os valores das chaves `codigo`

e `nome` são associados diretamente aos respectivos atributos, mas o valor da chave `codigo_setor` é associado ao atributo `codigo` do objeto da classe `Setor`, encapsulado pela instância de `Lanterna`.

Em seguida, adicione o método `getInputFilter()`:

```
    /**
     *
     * @return \Laminas\InputFilter\InputFilterInterface
     */
    public function getInputFilter()
    {
        if (! $this->inputFilter) {
            $inputFilter = new InputFilter();
            $factory = new InputFactory();
            $inputFilter->add($factory->createInput([
                'name' => 'nome',
                'required' => true,
                'filters' => [
                    [
                        'name' => 'StripTags'
                    ],
                    [
                        'name' => 'StringTrim'
                    ]
                ],
                'validators' => [
                    [
                        'name' => 'StringLength',
                        'options' => [
                            'encoding' => 'UTF-8',
                            'min' => 2,
                            'max' => 30
                        ]
                    ]
                ]
            ]));
            $inputFilter->add($factory->createInput([
                'name' => 'codigo_setor',
                'required' => true,
                'filters' => [
```

```
                [
                    'name' => 'Int'
                ]
            ],
            'validators' => [
                [
                    'name' => 'Digits'
                ]
            ]
        ]));
        $this->inputFilter = $inputFilter;
    }
    return $this->inputFilter;
}
```

Nesse método, adicionamos o filtro `StripTags` e `StringTrim` para o campo `nome`. O `StripTags` remove todas as tags do conteúdo, evitando injeções de HTML, JavaScript ou qualquer outra linguagem de marcação.

Já o filtro `StringTrim` remove espaços em branco à esquerda e à direita da expressão. O campo `nome` tem um único validador, `StringLength`, configurado para limitar o campo a cadeias de caracteres (com comprimento entre 2 e 30).

O campo `codigo_setor`, por sua vez, usa: o filtro `Int` para garantir que o valor recebido seja um inteiro; e o validador `Digits`, que verifica se o conteúdo é composto apenas por dígitos de 0 a 9.

Finalmente, adicione o método `getArrayCopy()`:

```
/**
 *
 * @return array
 */
public function getArrayCopy()
{
    return [
        'codigo' => $this->codigo,
        'nome' => $this->nome,
```

```
            'codigo_setor' => $this->setor->codigo
        ];
    }
}
```

Esse método cria uma representação em array do objeto, que será usada na persistência de seu conteúdo para um registro de tabela. O próximo passo é criar o mapeador.

O mapeador terá métodos correspondentes às operações de manipulação de registros em bancos de dados: criar, alterar, excluir e pesquisar. As operações de criação e alteração ficarão encapsuladas no método `saveModel`. A operação de exclusão ficará dentro do método `deleteModel`. Então, as de pesquisa serão feitas pelos métodos `getModel` e `fetchAll`.

Vamos criar o arquivo `LanternaTable.php` na pasta `module\Tropa\src\Model`, passo a passo. Primeiro, adicionaremos os namespaces usados e a abertura da declaração da classe:

```
<?php
namespace Tropa\Model;

use Laminas\Db\TableGateway\TableGatewayInterface;
use Laminas\Db\Adapter\Driver\ResultInterface;
use Laminas\Db\Sql\Select;

class LanternaTable
{
```

A interface `TableGatewayInterface` já foi utilizada na classe `SetorTable`, com o mesmo propósito: garantir que o objeto recebido pelo construtor possua os métodos `insert()`, `update()`, `delete()` e `select()`.

A interface `ResultInterface` garante que os métodos de consulta da classe seguirão o padrão de objeto iterável, usado na classe `Setor`. A classe `Select` é utilizada aqui pela primeira vez, pois é necessário construir uma declaração SQL `SELECT` que faça uma junção de duas

tabelas – o que não é possível apenas com o método `select()` de `TableGateway`.

Em seguida, crie os atributos da classe:

```
/**
 *
 * @var TableGatewayInterface
 */
private $tableGateway;

/**
 *
 * @var string
 */
private $keyName = 'codigo';
```

O atributo `tableGateway` armazenará o objeto de manipulação da tabela. O atributo `keyName` define já na declaração qual é o nome da chave primária da tabela mapeada.

Em seguida, acrescente o método construtor:

```
/**
 *
 * @param TableGatewayInterface $tableGateway
 */
public function __construct(TableGatewayInterface $tableGateway)
{
    $this->tableGateway = $tableGateway;
}
```

O construtor da classe faz a associação do parâmetro recebido com o atributo `tableGateway`, garantindo que ele seja um objeto que implemente `TableGatewayInterface`.

Em seguida, adicione o método `fetchAll()`:

```
/**
 *
 * @return ResultInterface
```

```
     */
    public function fetchAll()
    {
        $select = new Select();
        $select->from('lanterna')
        ->columns(array('codigo','nome'))
        ->join(array('s'=>'setor'), 'lanterna.codigo_setor = s.codigo',
            array('setor'=>'nome'));
        $resultSet = $this->tableGateway->selectWith($select);
        return $resultSet;
    }
```

Nesse método, é criada uma instância de `Select`, uma classe que representa uma declaração SQL `SELECT`. Todas as cláusulas de `SELECT` estão disponíveis nessa classe na forma de métodos.

O método `from()` define a tabela que se deseja consultar. O método `columns()` serve para definir quais campos da tabela consultada serão retornados no resultado. Já o `join()` serve para definir um `INNER JOIN`, uma junção de registros que corresponde à intersecção de campos de duas tabelas.

O array recebido pelo método `join()` como argumento é composto por dois elementos. O primeiro define um apelido para a tabela `setor` (s). O segundo elemento define o critério de junção, que são os registros na tabela `lanterna` cujo campo `codigo_setor` possui correspondentes no campo `codigo` da tabela `setor`. A consulta com esse objeto `Select` é feita passando-o como argumento para o método `selectWith()` de `TableGateway`.

Em seguida, adicione o método `getModel()`:

```
    /**
     *
     * @param string $keyValue
     * @return Setor
     */
    public function getModel($keyValue)
    {
        $select = new Select();
```

```
        $select->from('lanterna')
        ->columns(array('codigo','nome','codigo_setor'))
        ->join(array('s'=>'setor'), 'lanterna.codigo_setor = s.codigo',
            array('setor'=>'nome'))
            ->where(array('lanterna.codigo' => $keyValue));
        $rowset = $this->tableGateway->selectWith($select);

        if ($rowset->count()>0){
            $row = $rowset->current();
        } else {
            $row = new Lanterna();
        }

        return $row;
    }
```

O método `getModel()` também cria um `SELECT` com `INNER JOIN`, mas limita o resultado à junção cujo campo `codigo` da tabela `lanterna` tem o valor da variável `keyValue`.

Em seguida, adicione o método `saveModel()`:

```
    /**
     *
     * @param Setor $setor
     */
    public function saveModel(Lanterna $lanterna)
    {
        $data = array(
            'nome' => $lanterna->nome,
            'codigo_setor' => $lanterna->setor->codigo
        );
        $codigo = $lanterna->codigo;
        if (empty($this->getModel($codigo)->codigo)) {
            $this->tableGateway->insert($data);
        } else {
            $this->tableGateway->update($data, array(
                'codigo' => $codigo
            ));
        }
    }
```

À semelhança do que foi feito na classe `SetorTable`, o método `saveModel()` encapsula as operações de inclusão e alteração. A diferença é que, neste caso, o valor do campo `codigo` não basta para diferenciar um registro novo de um existente.

Como o código não é gerado pela tabela, mas fornecido pelo usuário, é necessário consultar a tabela com o método `getModel()` para verificar a existência do registro – e aí decidir se é uma inclusão ou alteração.

Finalmente, adicione o método `deleteModel()`:

```
    /**
     *
     * @param mixed $keyValue
     */
    public function deleteModel($keyValue)
    {
        $this->tableGateway->delete(array(
            $this->keyName => $keyValue
        ));
    }
}
```

Esse método é uma fachada para o método `delete()` de `TableGateway`, que apaga um registro que atende à condição de igualdade especificada no array passado como argumento.

Você deve estar se perguntando onde está o relacionamento. A tabela `setor` tem um relacionamento de um para muitos com a tabela `lanterna`. Por consequência, uma instância da classe `Setor` pode se relacionar com muitas instâncias da classe `Lanterna`.

No modelo `Lanterna`, declaramos um atributo `setor` que encapsula uma instância da classe `Setor` no método `exchangeArray()`. Além disso, observe que os métodos `fetchAll()` e `getModel()` definem uma junção entre as tabelas. É assim que o relacionamento é definido.

ATENÇÃO

O método `selectWith()` precisa receber um objeto `Laminas\Db\Select` cujo nome de tabela case com o nome de tabela configurado para o objeto `Laminas\Db\TableGateway`.

8.6 Mapeando várias tabelas

Como já vimos anteriormente, o mapeamento entre a tabela e a classe-modelo é feito pela classe `ServiceManager`, e isso é configurado na chave `service_manager` do arquivo `module.config.php` do módulo `Tropa`. Então, edite esse arquivo `Module.php` e altere a chave para que fique assim:

module.config.php=>service_manager

```
    'service_manager' => [
            'factories' => [
                LanternaTable::class =>  function($sm) {
                    $tableGateway = $sm->get('LanternaTableGateway');
                    $table = new LanternaTable($tableGateway);
                    return $table;
                },
                'LanternaTableGateway' => function ($sm) {
                    $dbAdapter = $sm->get('Laminas\Db\Adapter');
                    $resultSetPrototype = new ResultSet();
                    $resultSetPrototype->setArrayObjectPrototype(new
Lanterna());
                    return new TableGateway('lanterna', $dbAdapter, null,
$resultSetPrototype);
                },
                SetorTable::class =>  function($sm) {
                    $tableGateway = $sm->get('SetorTableGateway');
                    $table = new SetorTable($tableGateway);
                    return $table;
                },
```

```
                'SetorTableGateway' => function ($sm) {
                    $dbAdapter = $sm->get('Laminas\Db\Adapter');
                    $resultSetPrototype = new ResultSet();
                    $resultSetPrototype->setArrayObjectPrototype(new
Setor());
                    return new TableGateway('setor', $dbAdapter, null,
$resultSetPrototype);
                },
            ],
    ]
```

Você já deve estar preocupado: se o método fica desse tamanho já com duas tabelas, imagine com vinte. Esse é um problema cuja solução vai além do MVC, e não nos preocuparemos com isso neste livro. Você deve se preocupar somente em verificar se os namespaces a seguir são importados no cabeçalho do arquivo `module.config.php`.

```
use Tropa\Model\Lanterna;
use Tropa\Model\LanternaTable;
```

8.7 Formulário com mapeador

Agora precisamos de um formulário para incluir e alterar Lanternas Verdes. Ao contrário do formulário criado anteriormente, esse precisará interagir com o mapeador do modelo `Lanterna`.

Em `module\Tropa\src\Form`, crie o arquivo `LanternaForm.php`, com o seguinte conteúdo:

LanternaForm.php

```
<?php
namespace Tropa\Form;

use Laminas\Form\Form;
use Tropa\Model\SetorTable;
```

```php
class Lanterna extends Form
{
    private $table;

    public function __construct($name = null, array $options = array()) {
        if (isset($options['table'])){
            $this->table = $options['table'];
        } else {
            throw new \Exception('Form requires SetorTable instance');
        }
        parent::__construct('lanterna');
        $this->setAttribute('method', 'post');
        $this->add(array(
            'name' => 'codigo',
            'type'  => 'hidden'
        ));
        $this->add(array(
            'name' => 'nome',
            'attributes' => array(
                'type'  => 'text',
                'autofocus' => 'autofocus'
            ),
            'options' => array(
                'label' => 'Nome',
            ),
        ));
        $this->add(array(
            'name' => 'codigo_setor',
            'type'  => 'select',
            'options' => array(
                'label' => 'Setor',
                'value_options' => $this->getValueOptions()
            ),
        ));
        $this->add(array(
            'name' => 'submit',
            'attributes' => array(
                'type'  => 'submit',
                'value' => 'Gravar',
                'id' => 'submitbutton',
```

```
            ),
        ));

    }

    /**
     *
     * @return SetorTable
     */
    private function getSetorTable() {
        return $this->table;
    }

    /**
     *
     * @return Generator
     */
    private function getValueOptions() {
        $valueOptions = array();
        $setores = $this->getSetorTable()->fetchAll();
        $options = array();
        foreach($setores as $setor) {
            $options[$setor->codigo] = $setor->nome;
        }
        return $options;
    }
}
```

Observe que não precisamos passar o elemento `type` dentro de um array quando não há mais nenhum atributo a ser configurado. Para que o método `getSetorTable()` funcione, é necessário que a instância de `SetorTable` seja injetada na instância da classe `Tropa\Form\Setor` pelo controlador.

Em relação à classe que gera o formulário de setores, temos as seguintes diferenças:

- O método `getSetorTable()`, para recuperar uma instância de `SetorTable`;

- O método `getValueOptions()`, usado no construtor para alimentar a instância de `Laminas\Form\Element\Select`;
- `Laminas\Form\Element\Select` possui um atributo `valueOptions` para armazenar os itens da lista de seleção.

Pronto, já temos a programação necessária para a geração dos formulários dinâmicos usados para o cadastro dos dados de setores espaciais e Lanternas Verdes. No próximo capítulo, vamos nos aprofundar na construção de visões com modelos relacionados.

Capítulo 9
Visão e controle com relacionamentos

Poder é toda a chance, seja ela qual for, de impor a própria vontade numa relação social, mesmo contra a relutância dos outros. – Max Weber

Neste capítulo, finalizaremos o projeto `corps1` completando a implementação das camadas de visão e controle. Inicialmente, implementamos as visões de um modelo que não possui relacionamento, que é o dos setores espaciais. Agora vamos criar as visões de um modelo que possui relacionamento, o dos Lanternas Verdes.

O que veremos aqui se aplica a qualquer negócio que possua uma lógica de mestre-detalhe, aquela na qual um registro está relacionado a vários outros que trazem detalhes sobre seus dados. Exemplos de lógica mestre-detalhe são um pedido de compra e seus produtos, ou um álbum e suas músicas.

9.1 Visões de um modelo relacionado

Agora criaremos as visões para o cadastro de Lanternas Verdes. Na pasta `module\Tropa\view\tropa`, crie a pasta `lanterna`, que vai recepcionar os arquivos referentes ao controlador `LanternaController`, o qual será criado mais adiante. Dentro dela, vamos criar dois arquivos que exibirão o formulário de inclusão e alteração de Lanternas Verdes e a página de listagem dos cadastrados.

O primeiro será `edit.phtml`, que terá o seguinte conteúdo:

lanterna\edit.phtml

```
<?php
$title = $this->title . ' o Lanterna Verde';
$this->headTitle($title);
?>
<h1><?=$this->escapeHtml($title)?></h1>
<?php
$form = $this->form;
$form->setAttribute('action', $this->url(
                    'tropa',
                     [
                         'action' => 'save',
                         'controller' => 'lanterna'
                    ]));
$form->prepare();

echo $this->form($form);
```

Conforme você deve suspeitar, esta será a visão de inclusão e alteração. Mas para chegar a essa visão, precisamos de uma outra que tenha o link para inclusão e a listagem de registros, com links para alteração. Por isso, o próximo arquivo a ser criado é `index.phtml`, com o seguinte conteúdo:

lanterna\index.phtml

```
<?php
$this->headScript()->prependFile($this->basePath() . '/js/delete.js',
'text/javascript');
$title = 'Lanternas Verdes ';
$this->headTitle($title);
?>
<h1><?=$this->escapeHtml($title)?></h1>
<p>
    <a href="<?=$this->url(
        'tropa',
         [
             'action'=>'edit',
             'controller'=>'lanterna'
            ]
        )?>">Novo Lanterna Verde</a>
</p>
```

```
<table class="table">
<tr>
    <th>Código</th>
    <th>Nome</th>
    <th>Setor</th>
    <th> </th>
</tr>
<?php foreach ($this->models as $model):?>
<tr>
    <td><?=$model->codigo?></td>
    <td><?=$this->escapeHtml($model->nome)?></td>
    <td><?=$this->escapeHtml($model->setor->nome)?></td>
    <td>
        <a href="<?=$this->url('tropa',
            [
                'action'=>'edit',
                'controller'=>'lanterna',
                'key'  => $model->codigo
                ]
            )?>">Editar</a>
        <button class="delete" url="<?=$this->url('tropa',
            [
                'action'=>'delete',
                'controller'=>'lanterna',
                'key' => $model->codigo
                ]
            )?>">Remover</button>
    </td>
</tr>
<?php endforeach ?>
</table>
```

Observe que, com o método `escapeHtml()`, recuperamos o nome do setor a partir da instância de `Setor` encapsulada na instância de `Lanterna`, passando o atributo nome do objeto `Setor` como argumento. Um objeto dentro de outro objeto, cada um cuidando dos seus dados.

9.2 Um controlador com dois modelos pelo preço de um

Modelos e visões não devem ficar soltos como cavalos selvagens nas pradarias. Precisam de controle; ou, melhor, de controladores.

Anteriormente, criamos o controlador `SetorController` para gerenciar o cadastro de setores espaciais. Agora vamos criar o controlador `LanternaController` para gerenciar o cadastro de Lanternas Verdes. Ele será criado no arquivo `LanternaController.php` na pasta `module\Tropa\src\Controller` e se constituirá do seguinte código-fonte:

LanternaController

```
<?php
namespace Tropa\Controller;

use Laminas\Mvc\Controller\AbstractActionController;
use Laminas\View\Model\ViewModel;
use Tropa\Form\Lanterna as LanternaForm;
use Tropa\Model\Lanterna;
use Laminas\Session\Container as SessionContainer;
use Laminas\I18n\Translator\Resources;
use Laminas\Mvc\I18n\Translator as MvcTranslator;
use Laminas\Validator\AbstractValidator;
use Laminas\I18n\Translator\Translator;

class LanternaController extends AbstractActionController
{
    private $table;
    private $parentTable;

    public function __construct($table, $parentTable, $sessionManager)
    {
        $this->table = $table;
        $this->parentTable = $parentTable;
        $sessionManager->start();
    }
```

```
    public function indexAction()
    {
        return new ViewModel(
            ['models' => $this->table->fetchAll()]
        );
    }

    /**
     * Action to add and change records
     */
    public function editAction()
    {
        $codigo = $this->params()->fromRoute('key', null);
        $lanterna = $this->table->getModel($codigo);
        $form = new LanternaForm('lanterna',['table' => $this-
>parentTable]);
        $form->get('submit')->setValue(
            empty($codigo) ? 'Cadastrar' : 'Alterar'
            );
        $sessionContainer = new SessionContainer();
        if (isset($sessionContainer->model)){
            $lanterna->exchangeArray($sessionContainer->model->toArray());
            unset($sessionContainer->model);
            $form->setInputFilter($lanterna->getInputFilter());
            $this->initValidatorTranslator();
            $form->bind($lanterna);
            $form->isValid();
        } else {
            $form->bind($lanterna);
        }
        return [
            'form' => $form,
            'title' => empty($codigo) ? 'Incluir' : 'Alterar'
        ];
    }

    /**
     * Action to save a record
     */
    public function saveAction()
    {
```

```
        $request = $this->getRequest();
        if ($request->isPost()) {
            $form = new LanternaForm('lanterna',['table'=>$this-
>parentTable]);
            $lanterna = new Lanterna();
            $form->setInputFilter($lanterna->getInputFilter());
            $post = $request->getPost();
            $form->setData($post);
            if (! $form->isValid()) {
                $sessionContainer = new SessionContainer();
                $sessionContainer->model = $post;
                return $this->redirect()->toRoute('tropa', [
                    'action' => 'edit',
                    'controller' => 'lanterna'
                ]);
            }
            $lanterna->exchangeArray($form->getData());
            $this->table->saveModel($lanterna);
        }
        return $this->redirect()->toRoute('tropa', [
            'controller' => 'lanterna'
        ]);
    }

    /**
     * Action to remove records
     */
    public function deleteAction()
    {
        $codigo = $this->params()->fromRoute('key', null);
        $this->table->deleteModel($codigo);
        return $this->redirect()->toRoute(
            'tropa',
            ['controller'=>'lanterna']
            );
    }

    protected function initValidatorTranslator()
    {
        $translator = new Translator();
        $mvcTranslator = new MvcTranslator($translator);
```

```
        $mvcTranslator->addTranslationFilePattern(
            'phparray',
            Resources::getBasePath(),
            Resources::getPatternForValidator()
        );

        AbstractValidator::setDefaultTranslator($mvcTranslator);
    }
}
```

Observe que o controlador `LanternaController` possui um método construtor com dois argumentos não opcionais. Isso significa que o objeto dessa classe exigirá a passagem desses argumentos. Como não criamos diretamente o objeto, precisamos instruir o framework a fazer a injeção deles – `table` , `parentTable` e `sessionManager` – no controlador.

Abra o arquivo `module.config.php` . Se não sabe onde ele está, é só usar a combinação de teclas `CTRL+SHIFT+R` para localizar um arquivo pelo seu nome no Eclipse.

Faremos alterações na chave `controllers` do array retornado pelo arquivo `module.config.php` . As alterações consistem na definição de uma fábrica para a classe `LanternaController` , que vai criar objetos dessa classe com a injeção dos argumentos necessários. Isso será feito por meio de uma função anônima, executada quando o objeto da classe for solicitado via gerenciador de serviços.

O arquivo `module.config.php` deverá ficar assim:

module.config.php=>controllers

```
    'controllers' => [
        'aliases' => [
            'setor' => Controller\SetorController::class,
            'lanterna' => Controller\LanternaController::class
        ],
        'factories' => [
            Controller\IndexController::class => InvokableFactory::class,
            Controller\SetorController::class => function($sm){
```

```
                $table = $sm->get(SetorTable::class);
                $sessionManager = new SessionManager();
                return new Controller\SetorController($table,
$sessionManager);
            },
            Controller\LanternaController::class => function($sm){
                $table = $sm->get(LanternaTable::class);
                $parentTable = $sm->get(SetorTable::class);
                $sessionManager = new SessionManager();
                return new Controller\LanternaController($table,
$parentTable, $sessionManager);
            }
        ],
    ],
```

Já podemos testar nosso controlador. Você pode invocar a URL http://localhost:8000/tropa/lanterna/edit, e terá como resultado uma página como a da figura a seguir.

Figura 9.1: Página de inclusão de Lanternas Verdes

Após incluir alguns membros da Tropa, você terá como resultado uma página como a da próxima figura. E não há erro na imagem. É por isso que Mogo nunca vai às reuniões. Você pode testar a edição

e a remoção, que vão funcionar de forma idêntica ao cadastro de setores.

localhost:8000/tropa/lanterna
Sistema de Controle da Tropa dos Lanternas Verdes
Lanternas Verdes
Novo Lanterna Verde
Código Nome Setor
1 Rot Lop Fan Profundezas Obsidianas Editar Remover
2 Sinestro Korugar Editar Remover
3 Mogo Profundezas Obsidianas Editar Remover
4 Hal Jordan Terra Editar Remover
© 2020 by Oa Sector 0 All rights reserved.

Figura 9.2: Página de listagem de Lanternas Verdes

9.3 Ajustando o módulo Application

O módulo `Application`, como módulo padrão, será a porta de entrada para o nosso módulo `Tropa`. Só precisamos alterar a visão padrão, o arquivo `index.phtml`. Este é o que se encontra em `module\Application\view\application\index`. Modifique-o de modo que fique como mostra a listagem a seguir:

Application\view\index\index.phtml

```
<?php
$title = 'Cadastros ';
$this->headTitle($title);
?>
```

```
<h1><?=$this->escapeHtml($title)?></h1>
<ul>
<li><a href="<?=$this->url('tropa',['controller'=>'lanterna'])?
>">Lanternas Verdes</a></li>
<li><a href="<?=$this->url('tropa',['controller'=>'setor'])?>">Setores
Espaciais</a></li>
</ul>
```

Assim, quando invocarmos http://localhost:8000, teremos uma página com links para os cadastros de Lanternas Verdes e setores espaciais, como mostra a figura a seguir.

Figura 9.3: Página inicial da aplicação corps1

Assim terminamos um projeto com dois cadastros relacionados entre si, cobrindo as funcionalidades mais recorrentes em sistemas de informação web. A seguir, apresentamos um guia de referência rápida para a implementação MVC do Laminas com a qual trabalhamos.

Capítulo 10
Guia de referência rápida do MVC do Laminas

Pode ser que nos guie uma ilusão; a consciência, porém, é que nos não guia. – Fernando Pessoa

10.1 Modelos

A figura a seguir representa a estrutura geral da camada de modelo em uma aplicação que implementa MVC com Laminas.

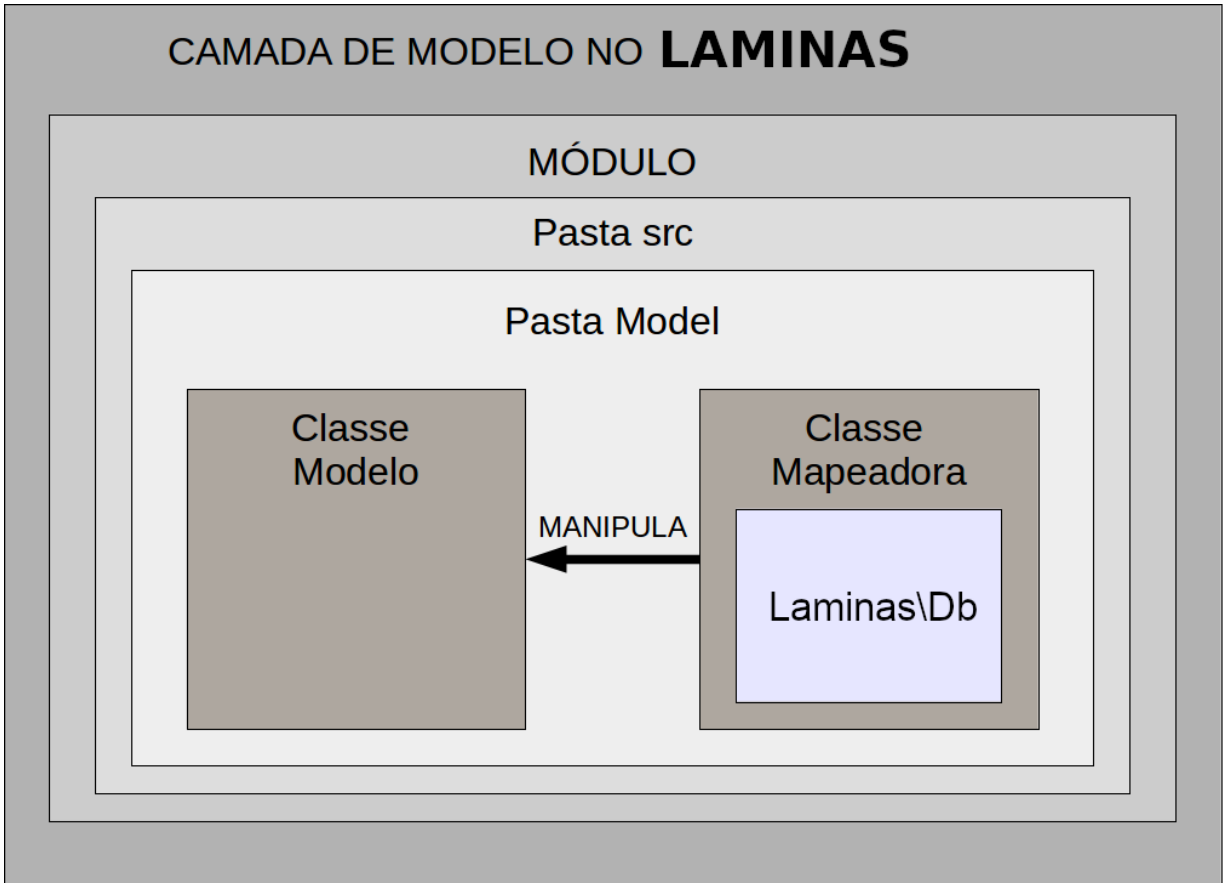


Figura 10.1: Camada de Modelo do módulo Laminas

Para implementar uma classe-modelo no Laminas, serão necessários, pelo menos, dois métodos: um para gravar os atributos e outro para lê-los.

exchangeArray()

```
public function exchangeArray($data)
```

Esse método é usado pela classe `Resultset` de `Laminas\Db\ResultSet` para popular os atributos do modelo. Nesse método, você recebe um array com os dados do modelo, inclusive os dados relacionados, e transfere-os para os atributos correspondentes.

getArrayCopy()

```
public function getArrayCopy()
```

Com este método, a classe `ResultSet` de `Laminas\Db\ResultSet` obtém um array com os atributos exclusivos do modelo. Ele também é utilizado por `Laminas\Form\Form` para configurar os valores para um formulário por meio do método `bind()`.

> **DICA**
>
> A classe `Laminas\Stdlib\ArrayObject` implementa os métodos `exchangeArray()` e `getArrayCopy()`.

10.2 Controladores

A próxima figura representa a estrutura geral da camada de controle em uma aplicação que implementa MVC com Laminas.

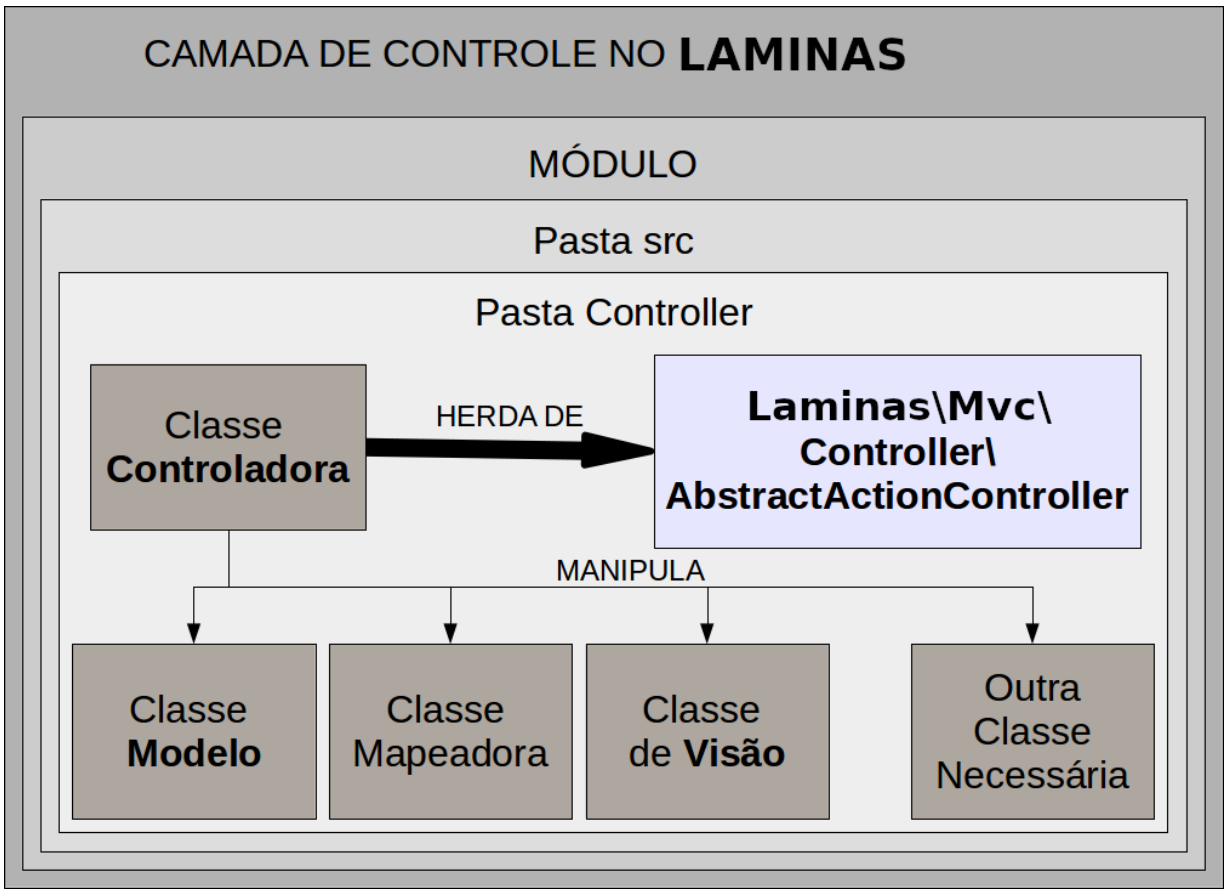


Figura 10.2: Camada de Controle do módulo Laminas

Visão geral

Controladores estendem a classe `Laminas\Mvc\Controller\AbstractActionController`, por padrão. Métodos que atendem a requisições terminam com o sufixo `Action`. Chamaremos esses métodos doravante de ações do controlador. Ações podem ter três tipos de retorno:

- Um objeto `ViewModel`;
- Um array;
- Um redirecionamento realizado pelo método `forward()` ou `redirect()`.

Se você tiver apenas de passar dados do controlador para a visão, use um array. Ele será injetado no construtor de um `ViewModel`.

O `AbstractActionController` possui um método `getRequest()` que retorna uma implementação de `Laminas\Stdlib\RequestInterface` – mais especificamente `Laminas\Http\PhpEnvironment\Request`. Esse objeto possui métodos úteis para identificar o conteúdo da requisição HTTP, dentre os quais:

- `getMethod()`: retorna o método da requisição (por exemplo, `DELETE`, `GET` ou `POST`);
- `getUri()`: retorna a URI como um objeto `Laminas\Http\Uri`;
- `getUriString()`: retorna a URI como texto;
- `getVersion()`: retorna a versão de HTTP;
- `getQuery()`: retorna o valor de um parâmetro enviado por HTTP `GET` (se o nome do parâmetro for omitido, retorna todos);
- `getPost()`: retorna o valor de um parâmetro enviado por HTTP `POST` (se o nome do parâmetro for omitido, retorna todos);
- `getCookie()`: retorna um objeto `Laminas\Http\Header\Cookie`;
- `isGet()`: retorna `true` se o método HTTP for `GET`;
- `isPost()`: retorna `true` se o método HTTP for `POST`;
- `isPut()`: retorna `true` se o método HTTP for `PUT`;
- `isDelete()`: retorna `true` se o método HTTP for `DELETE`;
- `isXmlHttpRequest()`: retorna `true` se a requisição for AJAX.

Existem classes auxiliares da camada de controle cujos métodos podem ser usados como se pertencessem aos próprios controladores, graças a uma arquitetura de injeção de dependência em tempo de execução. São os plugins dos controladores. Veremos os plugins disponíveis na camada de controle do Laminas a seguir.

Redirecionamento na mesma requisição

Para redirecionar o processamento para outra ação, na mesma requisição, utilize o plugin Redirect, assim:

```
$this->redirect()->dispatch('route', array('action' => 'method'));
```

O segundo argumento de `dispatch()` é um array com os parâmetros da rota. Se ele for omitido, o redirecionamento será feito para o

método `indexAction()` do controlador roteado.

O retorno do método `dispatch()` deve ser o retorno da ação redirecionadora.

Redirecionamento para outra requisição

Para redirecionar o processamento para outra ação, em uma nova requisição, utilize novamente o plugin `redirect` , assim:

```
$this->redirect()->redirect()->toRoute('route', array('action' => 'method'));
```

O segundo argumento de `toRoute()` é um array com os parâmetros da rota. Se ele for omitido, o redirecionamento será feito para o método `indexAction()` do controlador roteado.

O retorno do método `toRoute()` deve ser o retorno da ação redirecionadora.

Alteração de layout

Você pode alterar o layout de uma página usando o plugin `layout` , assim:

```
$this->layout()->setTemplate('layout/newlayout');
```

Considere que o caminho refere-se ao diretório `view` do módulo. Se você tiver preguiça, pode escrever desta forma:

```
$this->layout('layout/newlayout');
```

Recuperação de parâmetros da requisição

Para recuperar os parâmetros passados pela requisição, você pode usar o plugin `param` :

```
$this->params()->fromRoute('param', $default);
```

O segundo argumento de `fromRoute()` é o valor assumido se o parâmetro não for encontrado na requisição. Se você tiver preguiça, pode escrever assim:

```
$this->params('param', $default);
```

Geração de URLs

Para gerar URLs a partir de rotas, para enviar para a camada de visão, use o plugin `url` :

```
$this->url()->fromRoute('route', $params);
```

Recuperação do evento que invocou o controlador

Para obter a instância de `Laminas\Mvc\MvcEvent` em um método de um controlador, faça a seguinte chamada:

```
$event = $this->getEvent();
```

Recuperação do gerenciador de aplicação pelo controlador

Para obter a instância de `Laminas\Mvc\Application` em um método de um controlador, faça a seguinte chamada:

```
$application = $this->getEvent()->getApplication();
```

Seleção do modelo de visão de acordo com o tipo de requisição

Para selecionar um tipo de resposta de acordo com o cabeçalho HTTP `accept` da requisição, crie um array mapeando os modelos de visão para os valores esperados de cabeçalho, como neste exemplo:

```
 $acceptCriteria = [
      'Laminas\View\Model\JsonModel' => [
         'application/json',
      ],
      'Laminas\View\Model\FeedModel' => [
         'application/rss+xml',
```

```
        ],
    ];
```

Para instanciar o modelo de visão, passe o array como argumento para o plugin `AcceptableViewModelSelector`:

```
$viewModel = $this->acceptableViewModelSelector($this->acceptCriteria);
```

Mensagens armazenadas em sessão

Para adicionar uma mensagem que esteja disponível durante toda a sessão, use o plugin `flashMessenger`, seguindo este formato:

```
$this->flashMessenger()->addMessenger('Sua mensagem');
```

As mensagens são acumuladas e, quando uma nova é adicionada, as demais permanecem. Para saber se existem mensagens armazenadas na sessão, use o método `hasMessages()` assim:

```
$this->flashMessenger()->hasMessages()
```

Para saber se foram adicionadas mensagens na requisição atual, utilize o método `hasCurrentMessages()`:

```
$this->flashMessenger()->hasCurrentMessages()
```

Ambos retornam `true` ou `false`. Portanto, devem ser usados em estruturas de decisão.

Para remover as mensagens da sessão, use o método `clearMessages()` assim:

```
$this->flashMessenger()->clearMessages();
```

Para remover apenas as mensagens da requisição, utilize o método `clearCurrentMessages()`:

```
$this->flashMessenger()->clearCurrentMessages();
```

Esse plugin não faz parte do componente `Laminas\Mvc`. Para usá-lo, é necessário adicionar a dependência `laminas/laminas-mvc-plugin-`

`flashmessenger` no `composer.json` (e executar o `composer update`, é claro).

10.3 Visões

A próxima figura representa a estrutura geral da camada de visão em uma aplicação que implementa MVC com Laminas.

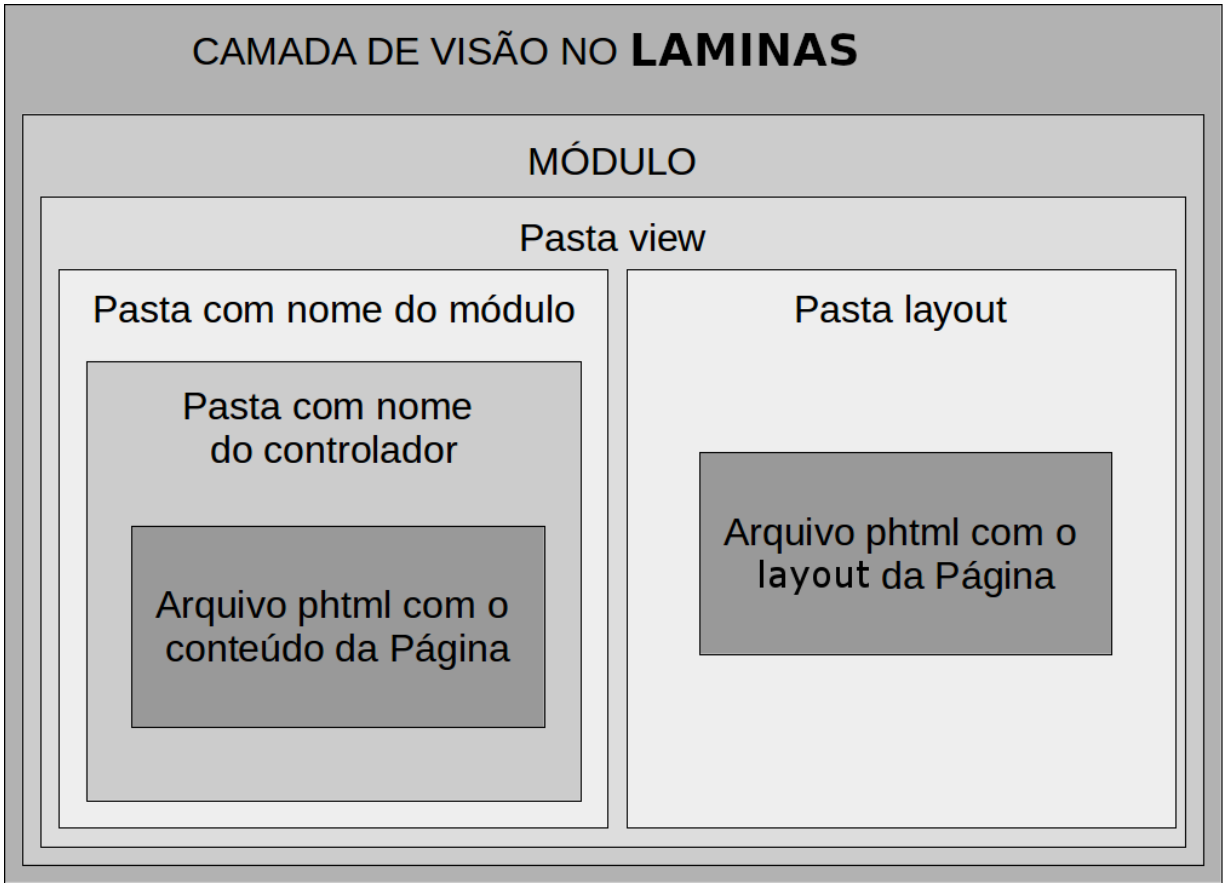

CAMADA DE VISÃO NO LAMINAS
MÓDULO
Pasta view
Pasta com nome do módulo
Pasta com nome do controlador
Arquivo phtml com o conteúdo da Página
Pasta layout
Arquivo phtml com o layout da Página

Figura 10.3: Camada de Visão do módulo Laminas

Para as subseções a seguir, vamos assumir que, quando o contexto indicar que haverá impressão de texto ou envio de saída para o

navegador, o método citado deverá ser passado como argumento para o comando `echo`, ou deverá ser usado o operador `=` da sintaxe curta do PHP.

Visão geral

Todos os elementos de array retornados pelas ações dos controladores ficam acessíveis nos arquivos `phtml` como atributos, referenciados como tais pela palavra-chave `$this`. Sendo assim, a ação pode retornar um valor desta forma:

```
return ['value'=>'alguma coisa'];
```

Ou desta forma, que é equivalente:

```
return new ViewModel(['value'=>'alguma coisa']);
```

Você ainda poderá criar o objeto `ViewModel` e associar valores com o método `setVariable()`, assim:

```
$viewModel = new ViewModel();
$viewModel->setVariable('value','alguma coisa');
return $viewModel;
```

O elemento do array será acessado no arquivo `phtml` desta forma:

```
$this->value
```

Se você quiser que o conteúdo HTML seja neutralizado, use o método `escapeHTML`. Isso garantirá também que acentos sejam exibidos corretamente, independente da codificação do navegador. O exemplo de código anterior seria escrito assim:

```
$this->escapeHtml(value)
```

Visão alternativa

Por padrão, uma ação renderiza um arquivo de mesmo nome. Assim, `indexAction()` renderiza o arquivo `index.phtml`; `addAction()`, o arquivo `add.phtml`; `editAction()`, o arquivo `edit.phtml`; e assim por

diante. Se você quiser que outro arquivo seja usado, utilize o método `setTemplate()` de `ViewModel`, assim:

```
$view = new ViewModel();
$view->setTemplate('pasta/outrapasta/arquivo');
return $view;
```

Não precisa colocar a extensão `phtml`, isso já é assumido. O caminho faz referência ao diretório `view` do módulo. O último nome sempre é o arquivo, por isso a quantidade de pastas não é determinada.

Visões compostas

Você pode agregar conteúdo de vários arquivos em uma única visão usando o método `addChild()` de `ViewModel`.

O método `addChild()` recebe dois argumentos, respectivamente, uma instância de ViewModel e um texto identificador, que será um atributo na visão principal.

```
$mainView = new ViewModel();
$headerView = new ViewModel();
$headerView->setTemplate('content/header');
$footerView = new ViewModel();
$footerView->setTemplate('content/footer');
$mainView->addChild($headerView, 'header') ->addChild($footerView,
'footer');
return $mainView;
```

No arquivo `phtml` principal, o conteúdo das outras páginas será referenciado assim:

```
<?=$this->header>
```

Veja o conteúdo principal:

```
<?=$this->footer>
```

Conforme vimos na seção anterior, o caminho pelo qual as páginas são procuradas inicia-se na pasta `view` do módulo.

Geração de URL

A camada de visão faz uso de classes auxiliares, cujas instâncias são injetadas em tempo de execução. Para criar URLs, usamos a seguinte interface:

```
$this->url('rota', ['action' => 'method', 'param' => $value])
```

Geração de listas

Para criar uma lista HTML, utilizamos a classe auxiliar `HtmlList`. Os itens são definidos por um array, passado como primeiro argumento para o método `htmlList()`. Por padrão, é criada uma lista não ordenada (`<UL></UL>`). Para criar uma lista ordenada (`<LI></LI>`), passamos `true` como segundo argumento.

```
// $items é um array
// lista não ordenada
$this->htmlList($items);
// lista ordenada
$this->htmlList($items, true);
```

Para configurar atributos para a lista, passamos um terceiro argumento para o método `htmlList()`, um array associativo, no qual as chaves são os nomes dos atributos.

```
// $attribs é um array
// lista não ordenada com atributos
$this->htmlList($items, false, $attribs);
// lista ordenada
$this->htmlList($items, true, $attribs);
```

Os itens da lista são neutralizados por padrão. Isso quer dizer que tags HTML serão convertidas para não serem interpretadas. Se você quiser desabilitar a neutralização, passe `false` como quarto argumento de `htmlList()`.

```
// lista não ordenada sem atributos sem neutralização de HTML
$this->htmlList($items, false, false, false);
// lista ordenada sem atributos sem neutralização de HTML
$this->htmlList($items, true, false, false);
```

Base do URL

Você pode obter o caminho base da aplicação, necessário para fazer referências a arquivos CSS e JavaScript, com a auxiliar `BasePath`. Para obter somente o caminho, pode chamar o método `basePath()` sem argumentos.

```
$this->basePath()
```

Para retornar um caminho relativo prefixado com o caminho base, passe o caminho relativo como argumento para o método `basePath()`, como no exemplo a seguir.

```
$this->basePath('css/main.css')
```

Não se esqueça de que a reescrita de URL afeta toda a aplicação. Isso significa que não basta tentar se referir ao arquivo, é necessário que a pasta em que ele se encontra desabilite o módulo de reescrita. Para fazer isso, use a instrução:

```
RewriteEngine off
```

Essa instrução pode estar em um arquivo `.htaccess` no diretório que será exceção à regra. Assim, as requisições HTTP a esse diretório serão tratadas normalmente, sem intermediação do Laminas.

Alternância de valores

Para alternar valores em iterações sobre coleções, podemos usar a auxiliar Cycle. Ela é usada desta forma:

```
$this->cycle(["valor1", "valor2"])->next()
```

O método `next()` retorna um dos valores do array, alternando-os a cada iteração do laço de repetição onde ele estiver contido. Você

pode ter vários ciclos em um mesmo laço de repetição, desde que dê nome a eles. Veja um exemplo:

```
$this->cycle(["valor1","valor2"],'ciclo1')->next()
$this->cycle(["outrovalor1","outrovalor2"],'ciclo2')->next()
```

Visão parcial

Você pode criar visões parciais parametrizadas com a auxiliar `Partial`. O método `partial()` recebe como argumentos um nome de arquivo (com extensão) e um array com parâmetros que serão tratados na versão parcial como atributos, assim como os valores passados por array, ou `ViewModel` do controlador para a visão. O formato geral do método `partial()` é o seguinte:

```
$this->partial('partial.phtml', [
    'atributo1' => 'valor1',
    'atributo2' => 'valor2']);
```

O arquivo da visão parcial é procurado no diretório `view`. O arquivo `partial.phtml` do exemplo utiliza os parâmetros desta forma:

```
<?=$this->atributo1?>
<?=$this->atributo2?>
```

Também é possível passar um objeto como segundo argumento de `partial()`. Se ele não for uma instância de uma classe que implementa o método `toArray()`, a função `object_get_vars()` será usada para retornar um array com todos os atributos públicos da classe.

Quando utilizar `Paginator` em conjunto com `PartialLoop`, você precisa usar o método `setObjectKey()`, definindo um nome pelo qual o objeto será referido na visão. Veja um exemplo:

```
$this->setObjectKey('modelo');
```

Neste caso, os atributos do objeto serão referidos a partir do nome definido, como no exemplo a seguir:

```
<?=$this->modelo->atributo1?>
```

Visão parcial de coleções

Para criar visões parciais que sejam repetidas para uma coleção de dados, que pode ser uma coleção de objetos ou um array, você pode usar a auxiliar `PartialLoop`. O método `partialLoop()` tem assinatura similar a `partial()`. A diferença está no segundo argumento, que obrigatoriamente deve ser uma coleção que atenda às seguintes regras:

- Ser um objeto de uma classe que implemente `Iterator`;
- Ser um array associativo de duas dimensões.

Persistência de conteúdo entre scripts

Para transmitir valores do controlador para a visão, usamos `ViewModel`, mas para transmitir valores do controlador para as auxiliares ou entre auxiliares dentro da mesma requisição, temos uma auxiliar específica, chamada `Placeholder`.

Para armazenar um valor com essa auxiliar, usamos o seguinte formato:

```
$this->placeholder('identificador')->set("valor")
```

Para recuperar um valor, usamos o método `placeholder()`, utilizando o mesmo identificador usado para armazená-lo. A recuperação do valor armazenado acima seria feita da seguinte forma:

```
$this->placeholder('identificador')
```

A partir do método `placeholder()`, podemos chamar os seguintes métodos:

- `setPrefix()`: configura um prefixo para o conteúdo armazenado;
- `getPrefix()`: recupera o prefixo configurado pelo método anterior;
- `setPostfix()`: configura um sufixo para o conteúdo armazenado;

- `getPostfix()` : recupera o sufixo configurado pelo método anterior;
- `setSeparator()` : configura um separador para elementos se o valor for um array;
- `getSeparator()` : recupera o separador configurado pelo método anterior;
- `setIndent()` : configura o número de espaços a ser usado na endentação;
- `getIndent()` : recupera o número de espaços configurado pelo método anterior.

Existem algumas implementações de `Placeholder` prontinhas para conteúdos de elementos de página HTML, que veremos a seguir.

10.4 Tipo de documento

Para gerar a tag `<!DOCTYPE>` , você pode utilizar a auxiliar `doctype()` . Esse método aceita os seguintes valores como argumento: `XHTML11` , `XHTML1_STRICT` , `XHTML1_TRANSITIONAL` , `XHTML1_FRAMESET` , `XHTML1_RDFA` , `XHTML1_RDFA11` , `XHTML5` , `XHTML_BASIC1` , `HTML4_STRICT` , `HTML4_LOOSE` , `HTML4_FRAMESET` , `HTML5` , `CUSTOM_XHTML` e `CUSTOM` .

A tag completa é impressa na visão desta forma:

```
$this->doctype()
```

Para obter o tipo de documento configurado, fazemos assim:

```
$this->doctype()->getDoctype()
```

Além disso, a partir de `$this->doctype()` , é possível chamar os métodos: `isXhtml()` , `isHtml5()` e `isRdfa()` .

Configuração da tag LINK

Para gerar a tag `<LINK>`, usamos o método `headLink`, que recebe como argumento um array com os atributos da tag. Para o caso específico de acrescentar folhas de estilo, usamos o método `appendStylesheet()`, chamado a partir de `headLink()`. Ele recebe como argumento o caminho para o arquivo CSS, relativo ao caminho base da aplicação.

A seguir, temos um exemplo de uso, criando tags para o arquivo `favicon.ico` (ícone usado pelo navegador quando carrega a página) e para uma folha de estilo. O segundo argumento de `headLink()` indica que o link será colocado antes dos demais.

```
$this->headLink(array(
    'rel'  => 'favicon',
    'href' => '/img/favicon.ico',
), 'PREPEND')
    ->appendStylesheet('/styles/basic.css')
    );
```

Também estão disponíveis os métodos `prependStylesheet()` e `setStylesheet()`. O primeiro acrescenta uma ligação com o arquivo CSS antes das demais. O segundo configura a tag LINK como uma ligação para uma folha de estilo.

Ou seja, se você quiser apenas criar uma ligação com uma folha de estilo, use `setStylesheet()`. Se quiser acrescentar uma ligação antes de outras, use `prependStylesheet()`. Se quiser acrescentar uma ligação ao final de outras, use `appendStylesheet()`.

Configuração da tag META

Para gerar a tag `<META>`, usamos a auxiliar `headMeta()`. A partir desse método, podemos chamar vários outros métodos:

appendName()

```
appendName($keyValue, $content, $conditionalName)
```

Pode ser usado, por exemplo, para configurar palavras-chaves utilizadas pelos mecanismos de busca.

```
$this->headMeta()->appendName('keywords', 'Laminas, framework, PHP');
```

Para configurar atributos HTTP para a página, usamos o método `appendHttEquiv()`. Por exemplo, para configurar o horário de expiração da página:

```
$this->headMeta()->appendHttpEquiv('expires','Thu, 5 Mar 2013 09:22:58 GMT');
```

Ou para configurar o tipo de conteúdo da resposta HTTP:

```
$this->headMeta()->appendHttpEquiv('Content-Type','text/html; charset=UTF-8');
```

Ou para configurar o idioma e a região:

```
$this->headMeta()->appendHttpEquiv('Content-Language', 'pt-BR');
```

setProperty()

```
setProperty($property, $content, $modifiers)
```

Para configurar propriedades para a tag `<META>`, usamos o método `setProperty()`. Esse método recebe dois argumentos essenciais. O primeiro representa o atributo `property`, e o segundo, o atributo `content`.

Configuração da tag SCRIPT

Você pode adicionar scripts ou importar arquivos com scripts usando a auxiliar `headScript`. Para adicionar um script diretamente na página, usamos o método `appendScript()`, que recebe o texto do script. Veja um exemplo:

```
$this->headScript()->appendScript($script);
```

Para adicionar uma importação de arquivo com script, usamos o método `appendFile()`, que recebe o caminho relativo ao arquivo que

contém o script. Por exemplo:

```
$this->headScript()->appendFile('/js/script.js');
```

Configuração da tag STYLE

Com a tag `<LINK>`, podemos importar arquivos de folhas de estilo em cascata (CSS). Mas você pode definir estilos diretamente na página com a tag `<STYLE>`. Para gerá-la com a classe `HeadStyle`, temos os seguintes métodos:

- `appendStyle($content, $attributes = array())` : acrescenta um estilo depois dos demais;
- `offsetSetStyle($index, $content, $attributes = array())` : define um estilo em uma posição determinada entre os demais;
- `prependStyle($content, $attributes = array())` : acrescenta um estilo antes dos demais;
- `setStyle($content, $attributes = array())` : configura um estilo. O argumento `$content` é o texto com a definição do estilo na sintaxe CSS. O array `$attributes`, que é opcional, contém os atributos da tag `<STYLE>`.

Configuração da tag TITLE

Para configurar e recuperar o título da página, usamos a auxiliar `headTitle`. Para definir um título, fazemos assim:

```
$this->headTitle('título');
```

As chamadas a `headTitle()` são cumulativas. Cada chamada concatena o texto ao que foi definido anteriormente, na mesma requisição. Para recuperar o título definido, chamamos o mesmo método sem argumentos:

```
$this->headTitle();
```

Configuração de objetos HTML embutidos

Há quatro métodos auxiliares para embutir objetos em páginas HTML:

- `htmlFlash()` : gera tags para embutir arquivos Flash;
- `htmlObject()` : gera tags para embutir um objeto customizado;
- `htmlPage()` : gera tags para embutir outras páginas (X)HTML;
- `htmlQuicktime()` : gera tags para embutir arquivos QuickTime.

O primeiro argumento, obrigatório para todos esses métodos, é o caminho relativo ao arquivo a ser carregado pela página.

Conteúdo JSON

O método `json()` mata vários coelhos com uma cajadada só. Ele configura a página para o formato JSON, desabilita o layout e converte os dados para o formato JSON. Seu uso é bastante simples:

```
$this->json($this->data)
```

O `$this->data` representa os dados passados pelo controlador.

10.5 Mapeadores

A classe `Laminas\Db\TableGateway` precisa receber em seu construtor os seguintes argumentos, nesta ordem:

- Nome da tabela mapeada;
- Objeto de classe que implementa `AdapterInterface` , responsável pela conexão com o banco de dados;
- Objeto que define características do mapeador – é um argumento opcional e pode receber `null` ;
- Objeto de classe que implementa `ResultSetInterface` , que encapsulará os registros da tabela.

A classe `TableGateway` dispõe dos seguintes métodos:

- `delete(Where\| Closure| string| array $where)` , que apaga um registro;
- `insert(array $set)` , que inclui um registro;
- `select(Where\| Closure| string| array $where)` , que recupera um conjunto de registros;
- `update(array $set, | string| array| closure $where)` , que atualiza um conjunto de registros.

10.6 Formulários

Para definir elementos para um formulário em uma classe que estende `Laminas\Form\Form` , há duas notações. A seguir, mostramos exemplos de como criar uma caixa de texto, uma caixa de seleção e um botão de submissão usando primeiro a definição com array, e depois com objeto:

Array

```
$nome = [
            'name' => 'nome',
            'type'  => 'text',
            'options' => [
                    'label' => 'Nome',
            ]
    ];
    $this->add($nome);
    $codigoSetor = [
            'name' => 'codigo_setor',
            'type'  => 'select',
            'options' => [
                    'label' => 'Setor',
                    'value_options' => $this->getValueOptions()
            ]
    ];
    $this->add($codigoSetor);
    $submit = [
```

```
            'name' => 'submit',
            'attributes' => [
                    'type'  => 'submit',
                    'value' => 'Gravar',
                    'id' => 'submitbutton',
            ],
    ];
    $this->add($submit);
```

Objeto

```
// Text está no namespace Laminas\Form\Element
$nome = new Text('nome');
$nome->setLabel('Nome');
$this->add($nome);
// Select está no namespace Laminas\Form\Element
$codigoSetor = new Select('codigo_setor');
$codigoSetor->setLabel('Setor')
                ->setValueOptions($options);
    $this->add($codigoSetor);
// Submit está no namespace Laminas\Form\Element
$submit = new Submit('submit');
$submit->setLabel('Gravar') ->setAttribute('id', 'submitbutton');

$this->add($submit);
```

Os métodos comuns para as instâncias do namespace `Laminas\Form\Element` são:

- `setAttribute()` , que configura um atributo do elemento;
- `setAttributes()` , que configura um conjunto de atributos e recebe um array;
- `setLabel()` , que configura o rótulo a ser exibido antes do elemento;
- `setOptions()` , que configura opções que não sejam atributos do elemento;
- `setValue()` , que configura o valor contido ou selecionado pelo elemento.

A seguir, vamos listar as classes que geram elementos de formulário com seus principais métodos específicos quando for o caso. Na maior parte dos casos, utiliza-se o método `setAttributes()` para configurar o objeto que define o elemento de formulário.

Elementos-padrão

O namespace `Laminas\Form\Element` contém classes que renderizam elementos HTML. As classes que implementam a mesma interface de métodos são: `Button`, `Csrf`, `File`, `Hidden`, `Image`, `Password`, `Submit`, `Text` e `Textarea`.

As classes que possuem métodos específicos são:

Captcha

Método específico: `setCaptcha()`.

Checkbox

Métodos específicos:

- `setUseHiddenElement()`
- `setCheckedValue()`
- `setUncheckedValue()`

Collection

Métodos específicos:

- `setCount()`
- `setTargetElement()`
- `setShouldCreateTemplate()`

MultiCheckbox

Método específico: `setValueOptions()`.

Radio

Método específico: `setValueOptions()`.

Select

Método específico: `setValueOptions()`.

Elementos HTML5

Laminas possui renderizadores para elementos HTML5: `Color`, `Date`, `DateTime`, `DateTimeLocal`, `Email`, `Month`, `MonthSelect`, `Number`, `Range`, `Time`, `Url` e `Week`.

10.7 Aplicação

Alteração da URL base da aplicação

A URL base afeta o roteamento da aplicação. A partir de uma instância de `Laminas\Mvc\Application`, é possível ter acesso ao objeto que encapsula a requisição, que, por sua vez, possui o método `setBaseUrl()`.

```
$application->getRequest()->setBaseUrl([URL BASE]);
```

Alteração do caminho base da aplicação

O caminho base afeta a interface com o usuário, ao determinar o local de origem de arquivos CSS, JavaScript e outros usados pelas páginas HTML. A partir de uma instância de `Laminas\Mvc\Application`, é possível ter acesso ao objeto que encapsula a requisição, que, por sua vez, possui o método `setBasePath()`.

```
$application->getRequest()->setBasePath([CAMINHO BASE]);
```

10.8 Fluxo de processamento da requisição

A requisição HTTP submetida à aplicação é direcionada ao arquivo `index.php` por força do redirecionamento determinado pelo módulo de reescrita de URL. No arquivo `index.php`, ocorre a inversão de controle da aplicação, que passa para o objeto `Laminas\Mvc\Application`. Este lê o arquivo de configuração passado como argumento para o método `init()` e, a partir dele, instancia o `ServiceManager` e o `ModuleManager`, e inicializa os ouvintes registrados.

O método `run()` de `Application` dispara o evento `ROUTING`, durante o casamento da URL com as rotas configuradas pelo módulo. Nesse momento, a aplicação descobre qual exatamente é a ação solicitada. A ação será efetivamente invocada no próximo evento disparado, o `DISPATCH`.

Nesse evento, o objeto de requisição é delegado a um controlador, que o processa e gera um resultado visual a partir de um arquivo `phtml`. Esse resultado será combinado a um arquivo de layout, e o resultado completo será enviado como resposta para o cliente solicitante, provavelmente em um navegador de internet.

A ação do controlador vai processar os dados da requisição utilizando os objetos modelo, que contêm as regras de negócio da aplicação. O tratamento dos dados para a apresentação é feito com as classes `view helpers`.

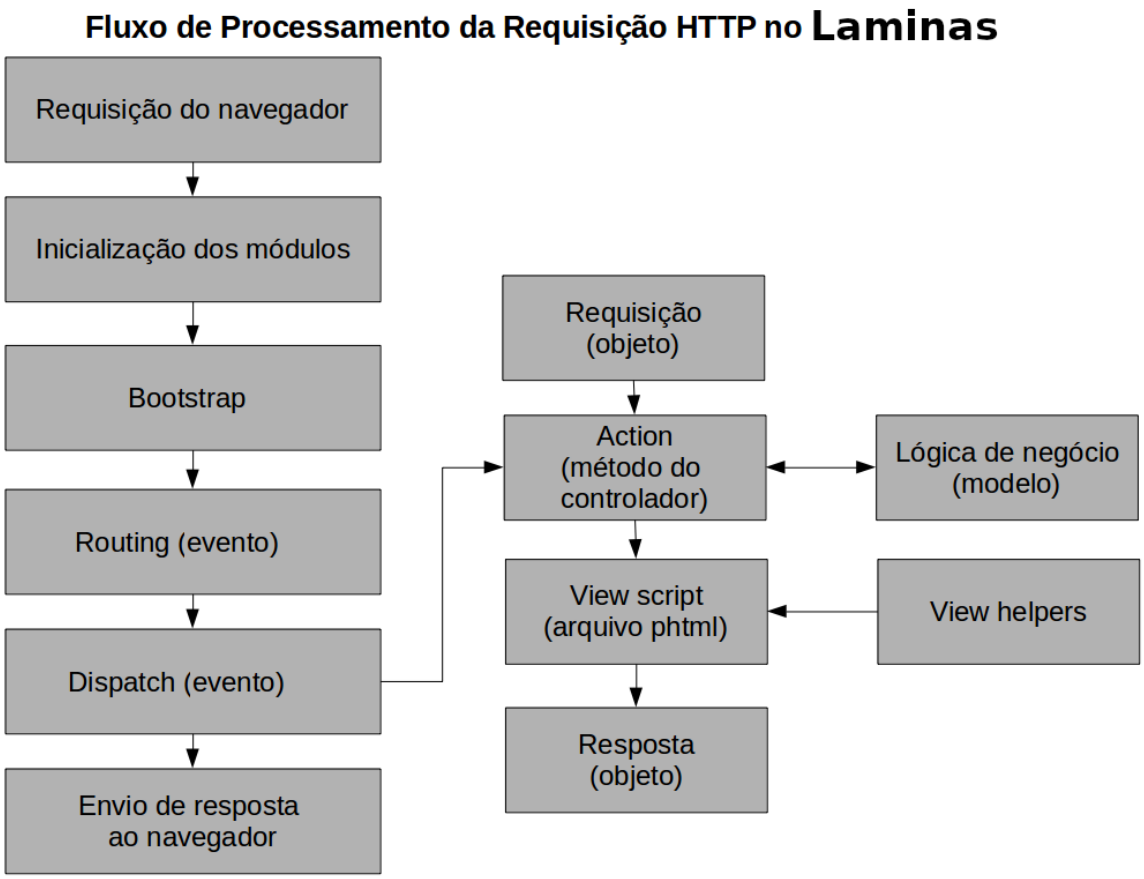


Figura 10.4: Fluxo de processamento da requisição

Capítulo 11
Considerações finais

É estranho, mas as coisas boas e os dias agradáveis são narrados depressa. – J. R. R. Tolkien

O capítulo final sempre tem um tom de despedida, mas não podemos dar as mãos e caminhar em direção ao pôr do sol antes de apresentar algumas informações que vão complementar e suplementar todo o conteúdo deste livro.

A fonte primária de informações sobre o Laminas é a documentação disponível no site do projeto. Embora ela se pareça muitas vezes com os enigmas do Mestre dos Magos, é a expressão daqueles que criaram e mantêm o software.

Além da documentação, você pode encontrar artigos sobre Laminas no blog https://getlaminas.org/blog e questões resolvidas no fórum https://discourse.laminas.dev.

Há vários componentes que fazem parte do Laminas e podem ser adicionados ao seu projeto conforme sua necessidade. Para saber quais são os disponíveis, acesse uma das seguintes páginas:

- https://packagist.org/packages/laminas/
- https://getlaminas.org/

Lembre-se de que o comando deve ser executado dentro do diretório do projeto. Porém, você pode usar o Composer para baixar um componente qualquer dentro de uma outra aplicação PHP que não use o MVC do Laminas, criando um `composer.json` com a especificação da dependência. Você também pode executar o Composer pelo Eclipse PDT, usando o plugin ou a configuração de programas externos.

A documentação do Laminas compreende mais de 60 componentes diferentes, sendo assim digna de uma literatura muito mais extensa. Entretanto, este livro seguramente abrange os principais tópicos da implementação MVC do Laminas de uma forma profunda, tratando dos fundamentos de organização de um sistema de informações orientado a objetos e eliminando sua preocupação com detalhes que não interessam ao cliente.

Espero que esta obra lhe estimule a prosseguir no estudo e no uso do Laminas. Este livro não teria sido realizado se eu tivesse esperado as condições perfeitas para sua realização. Como disse o general Aníbal quando questionado sobre o ataque a Roma com elefantes passando pelos Alpes: se não existir um caminho, nós abriremos um. Eventualmente, alguns elefantes ficam no caminho, e com isso quero dizer que você pode encontrar falhas. Se encontrar, por favor, envie uma errata.

Se tiver qualquer dúvida, crítica, sugestão ou informações sobre invasões alienígenas, pode enviar uma mensagem para livros@fgsl.eti.br. Muito obrigado por participar deste empreendimento, e espero que a leitura lhe estimule a participar dos próximos, que abordarão os componentes e serviços do Laminas. Boa sorte!

Capítulo 12
Referencial teórico

Neste capítulo, reunimos o referencial teórico relacionado a frameworks de desenvolvimento de software. A construção de um framework envolve uma grande bagagem de conhecimentos sobre programação orientada a objetos e arquitetura de software. Ao longo de anos, essa bagagem foi sendo acomodada no compartimento de carga do avião da Ciência da Computação.

Para tornar seu voo rumo a uma forma mais produtiva e organizada de desenvolver software web orientado a objetos, é importante que você compreenda as definições e os conceitos relacionados ao framework abordado neste livro. Aqui selecionamos autores cujas obras são parte fundamental da base teórica de frameworks de desenvolvimento.

Software

De acordo com Hoglund e McGraw (2006, p. 2), “o software é o que faz a diferença entre os computadores e as outras inovações tecnológicas”. Para definir o que é software, eles recorrem a Lady Ada Lovelace, que o prenunciou como uma “expressão imaterial de qualquer função indefinida em qualquer nível de generalidade e complexidade”.

Manutenção de software

Segundo Weinberg (apud PARISH, 1990, p. 1), “o principal problema na atividade de manutenção é que você simplesmente não pode fazer a manutenção em um sistema que não foi projetado para manutenção. A não ser que projetemos para manutenção, sempre teremos muitos problemas depois que um sistema entrar em produção”.

De acordo com Glass (2009), a manutenção de software é:

- Intelectualmente complexa, porque requer inovação enquanto coloca restrições severas à inovação;
- Tecnicamente difícil, uma vez que o mantenedor deve ser capaz de trabalhar com um conceito, um projeto e seu código, tudo ao mesmo tempo;
- Desleal, pois o mantenedor nunca consegue fazer todas as coisas que precisa, como gerar uma boa documentação da manutenção;
- Uma atividade sem vencedores, porque o mantenedor vê somente pessoas que têm problemas;
- Um trabalho sujo, já que o mantenedor deve trabalhar em um nível desmazelado de codificação detalhada;
- O mesmo que viver no passado, porque o código foi provavelmente escrito por outras pessoas antes que elas fossem boas nisso;
- Conservativa – o lema da manutenção é “se não está quebrado, não conserte”.

De acordo com Passos (2013):

- “Manutenções podem gerar problemas”;
- “Mudanças triviais podem gerar uma reação em cadeia de erros”;
- “Se o software é feito em grupo, os erros aumentam”.

Complexidade de software

Hoglund e McGraw (2006, p. 13) afirmam que “o software moderno é complicado, e as tendências sugerem que ficará ainda mais complicado em um futuro próximo”.

Lehman e Ramil (2001 apud HATTORI, 2008, p. 18) afirmam que “a evolução de software reflete-se numa necessidade intrínseca pelo desenvolvimento e a manutenção contínua de sistemas, para

endereçar uma aplicação, ou resolver um problema no domínio do mundo real".

Falhas em software

Passos (2013) afirma que "não somos capazes de escrever programas isentos de falhas".

Construção de software

Segundo McConnel (2005, p. 44), "a construção [de código] toma normalmente de 30% a 80% do tempo total [de desenvolvimento]".

Arquitetura de software

Arquitetura de software, segundo a norma IEEE 1471 (EELES, 2006), é a organização fundamental de um sistema incorporada pelos seus componentes, os relacionamentos entre eles e o ambiente, e os princípios que norteiam seu projeto e sua evolução.

Fowler (2006, p. 23) afirma que arquitetura "é um termo que muitas pessoas, com pouca concordância entre si, tentam definir". Mas ele concorda que há dois elementos em comum: a decomposição em alto nível de um sistema em suas partes e as decisões difíceis de alterar. Uma observação importante que Fowler (2006, p. 23) faz é que "a visão do que é significativo em termos de arquitetura pode mudar durante o ciclo de vida de um sistema".

Segundo Bass et al. (2003), a arquitetura de um sistema não se resume apenas a identificar quantos elementos o compõem e quais são as conexões entre eles. A arquitetura consiste em compreender qual é a natureza dos elementos, quais são suas responsabilidades, qual é o significado das conexões e qual é o significado da disposição dos elementos.

Para eles (BASS et al., 2003), a arquitetura de software de um programa ou sistema de computação é a estrutura ou as estruturas do sistema, o que compreende os elementos de software, as

propriedades desses elementos que são visíveis externamente e os relacionamentos entre eles.

A importância da arquitetura reside no fato de que, segundo Bosch (2000 apud Sommerville, 2007, p. 162), ela “afeta o desempenho, facilidade de distribuição e de manutenção de um sistema”. Bass et al. (2003) ainda acrescenta que a arquitetura de software:

- Representa uma abstração comum de um sistema que a maioria (se não todos) dos patrocinadores pode usar como uma base para uma mútua compreensão, negociação, consenso e comunicação;
- Manifesta o conjunto de decisões iniciais do projeto;
- E constitui um modelo relativamente pequeno e intelectualmente compreensível de como um sistema está estruturado e como seus elementos trabalham juntos.

Segundo Coplien (2010), a arquitetura é a essência da estrutura. Já Fowler identifica que decisões difíceis de mudar constituem um elemento da arquitetura de software. De modo mais específico, Bass et al. (2003) afirmam que as decisões iniciais são as mais difíceis de corrigir e mais complicadas para mudar posteriormente.

Segundo Sommerville (2007), uma das seis melhores práticas de engenharia de software, recomendadas pelo RUP (*Rational Unified Process*), é o uso de arquiteturas baseadas em componentes, justamente por envolver uma abordagem orientada a reúso.

Bass et al. (2003) citam três conceitos úteis que capturam elementos de uma arquitetura: o padrão arquitetural, o modelo de referência e a arquitetura de referência. Esses conceitos atuam como limitadores que inibem a diversidade na definição de arquiteturas.

Para Bass et al. (apud SOMMERVILLE, 2007, p. 162), um modelo de arquitetura de sistema promove o reúso em larga escala, pois “é uma descrição compacta e administrável de como um sistema está organizado e de como os componentes operam entre si”. Segundo

eles, “a arquitetura de sistema é muitas vezes a mesma para sistemas com requisitos similares e, assim, pode apoiar o reúso de software em larga escala”.

Reúso

Bass et al. (2003) afirmam que 80% do custo de um típico sistema de software ocorre depois da distribuição inicial. Paula Filho (2009, p. 256), por outro lado, afirma que, “quando uma organização começa a usar processos definidos de desenvolvimento de software, os maiores ganhos iniciais resultam da redução dos defeitos introduzidos em cada iteração”.

Ele explica que “isso ocorre por causa da redução do desperdício de tempo e dinheiro, principalmente aquele que é causado por defeitos de requisitos, análise e desenho”. Assim, “a partir daí, ganhos significativos só são conseguidos por meio de reutilização” (PAULA FILHO, 2009, p. 256).

McConnel (2005, p. 498) define reusabilidade como uma característica interna da qualidade de software, que se refere a quanto e com que facilidade um desenvolvedor pode usar partes de um sistema em outros.

Paula Filho (2009) afirma que há dois tipos de reutilização, a de código e a de desenho (projeto). Segundo esse autor, entre as formas de reutilização de código estão a de: objetos, quando módulos de código binário de interface padronizada são reaproveitados; classes, o reaproveitamento de classes fundamentais e de bibliotecas-padrão; e plataformas, que se refere ao reaproveitamento de camadas inteiras da arquitetura de uma aplicação.

Gamma et al. (2000, p. 39) afirmam que “a chave para maximização da reutilização está na antecipação de novos requisitos e mudanças nos requisitos existentes e em projetar sistemas de modo que eles possam evoluir de acordo”.

Sommerville (2007, p. 273), por sua vez, afirma que “o movimento para o desenvolvimento baseado em reúso foi uma resposta às demandas por menores custos de produção e manutenção de software, entregas mais rápidas de sistemas e aumento da qualidade do software”.

Sommerville ainda declara que o reúso sistemático deve ser planejado e introduzido por meio de um programa que envolva toda a organização. Segundo ele, as técnicas desenvolvidas para apoiar o reúso de software exploram “o fato de que sistemas em um mesmo domínio de aplicação são similares, que o reúso é possível em diferentes níveis (a partir de funções simples até aplicações completas) e que padrões para componentes reusáveis facilitam o reúso”.

De acordo com Parnas (apud BLOCH, 2006), “reúso é alguma coisa que é mais fácil falar do que fazer. Fazê-la requer tanto um bom projeto quanto uma boa documentação. Mesmo quando vemos um bom projeto, o que não é frequente, não veremos os componentes reusados sem uma boa documentação”.

Encapsulamento

Segundo Page-Jones (1997, p. 8), “encapsulamento é um conceito quase tão antigo quanto o próprio software. Desde a década de 1940, os programadores começaram a observar que padrões similares de instruções apareciam várias vezes dentro de um mesmo programa. Algumas pessoas (como Maurice Wilkes e seus colegas da Universidade de Cambridge) perceberam que esses padrões repetidos poderiam ser amontoados em um canto do programa e invocados através de um único nome a partir de vários pontos diferentes no programa principal”.

Padrões de projeto

Segundo Metsker (2004, p. 17), um padrão “é uma maneira de fazer algo, ou de buscar um objetivo. Em qualquer atividade que já esteja

madura ou em vias de amadurecer, encontraremos métodos eficazes comuns para atingir objetivos e para resolver problemas em vários contextos". Nesse entendimento, o autor define padrão de projeto como "a maneira de alcançar um objetivo – que utiliza classes e seus métodos em uma linguagem orientada a objetos".

Appleton (apud PRESSMAN, 2004, p. 191) define um padrão de projeto como "uma porção identificada de conhecimento profundo, que transmite a essência de uma solução provada, para um problema recorrente em certo contexto, em meio a preocupações concorrentes".

Gamma et al. (2000, p. 39) afirmam que os "padrões de projeto ajudam a evitar" problemas relacionados a mudanças "ao garantirem que o sistema possa mudar segundo maneiras específicas". De acordo com eles, "cada padrão de projeto permite a algum aspecto da estrutura do sistema variar independentemente de outros aspectos".

McConnell (2005, p. 133) descreve diversas vantagens que os padrões de projeto oferecem:

- Reduzem a complexidade, fornecendo abstrações prontas;
- Reduzem os erros, institucionalizando os detalhes de soluções comuns;
- Fornecem valor heurístico, sugerindo alternativas de design;
- Otimizam a comunicação, movendo o diálogo do projeto para um nível mais alto.

Metsker (2004, p. 23) resume padrões de projeto como "destilações de sabedoria acumulada" que fornecem "um jargão-padrão" e nomeiam "conceitos que praticantes experimentados aplicam". Ao utilizar padrões de projeto, evita-se explicações detalhadas de toda linha de código para outros programadores, de modo que eles entendam a estratégia de projeto encontrada em um código pela associação a um identificador comum. Além disso, "os padrões

podem acelerar as discussões de projeto, permitindo aos designers pensarem e discutirem em uma glanuralidade maior”.

McConnell (2005, p. 133) declara que, “como os padrões representam maneiras padronizadas de resolver problemas comuns, eles incorporam o conhecimento acumulado durante vários anos de tentativas de resolver esses problemas”. O mesmo autor ressalta que “percorrer um conjunto de alternativas familiares é muito mais fácil do que criar uma nova solução de projeto. E o código proveniente de um padrão conhecido também será mais fácil de ler do que um código totalmente personalizado”.

Dentre os padrões de projeto, Fowler destaca um conjunto que ele denomina de padrões de arquitetura. Segundo ele, esses padrões “direcionam o modo pelo qual a lógica de domínio se comunica com o banco de dados” (2006, p. 52).

Gamma et al. (2000, p. 17) afirmam que os padrões de projeto “resolvem problemas específicos”. Ao mesmo tempo, eles dizem que o projeto de software “deve ser específico para o problema a resolver, mas também genérico o suficiente para atender problemas e requisitos futuros”.

Camadas de software

Fowler dedica um capítulo inteiro à criação de camadas para projetos de software. De acordo com o autor, camadas são “os subsistemas principais no software dispostos de forma parecida com camadas de um bolo, em que cada camada repousa sobre uma camada mais baixa” (FOWLER, 2006, p. 37). Em sua definição, ele afirma que existe uma disciplina de consumo de serviços entre as camadas. Essa disciplina determina o seguinte:

- Cada camada se comunica somente com as camadas imediatamente abaixo e acima dela;
- Uma camada só pode consumir serviços da camada abaixo dela;

- Uma camada só pode oferecer serviços para uma camada acima dela;
- Como consequência, cada camada ignora a existência de outras camadas mais altas e mais baixas.

Sommerville (2007) cita o modelo de referência OSI como um exemplo de arquitetura em que cada camada depende somente da camada inferior a ela. Segundo ele, a vantagem dessa restrição é que, à medida que a tecnologia se desenvolvesse, cada camada "poderia ser novamente implementada de maneira transparente, sem afetar os sistemas que usassem as outras camadas" (SOMMERVILLE, 2007, p. 174).

Há um propósito para o uso de camadas em sistemas de software. Fowler (2006) destaca como benefícios:

- Compreensão de uma única camada como um todo coerente, sem a necessidade de saber muito sobre as demais camadas;
- Uma camada pode ser substituída por implementações alternativas dos mesmos serviços básicos;
- O uso de camadas diminui a dependência entre as partes de um sistema;
- Camadas são bons lugares para definir padrões;
- Uma vez construída, uma camada pode ser usada por muitos serviços de nível mais alto.

Contudo, Fowler (2006) afirma que o uso de camadas possui dois aspectos negativos. O primeiro refere-se ao fato de que as camadas encapsulam bem algumas coisas, mas não todas. Isso significa que determinadas alterações em um sistema implicam em modificações em todas as camadas.

O segundo aspecto refere-se ao desempenho da aplicação. A principal questão em si não é, como se pode pensar, a quantidade de camadas usadas, mas, sim, como ocorre a comunicação entre elas e como elas lidam com o gerenciamento de processos concorrentes.

Fowler (2006) identifica três camadas principais que podem ser empregadas em aplicações corporativas. Em sua própria definição, aplicações corporativas são aquelas que muitas vezes têm dados complexos para trabalhar, aliados a regras de negócio que não passam em nenhum teste de raciocínio lógico.

Para o contexto de aplicações corporativas, podemos considerar as camadas identificadas pelo autor. São elas: apresentação, domínio e fonte de dados.

- A apresentação é a camada que tem como função o fornecimento de serviços e a exibição de informações;
- O domínio contém a lógica, que é o real propósito do sistema;
- A fonte de dados refere-se à comunicação com bancos de dados, sistemas de mensagens, gerenciadores de transações e outros pacotes.

Fowler (2006) cita outros esquemas de camadas. Porém, analisando as responsabilidades das camadas desses esquemas, é fácil perceber que todos contêm as três que ele identificou. Pode-se assim dizer que estas constituem um padrão.

Fowler (2006) também cita um padrão de projeto cujas camadas têm uma grande similaridade com as definidas por ele como principais em uma aplicação corporativa: o MVC. O MVC nasceu como um framework para Smalltalk no final da década de 1970. Ainda segundo o autor, uma das principais vantagens desse padrão MVC é a independência que o modelo tem da apresentação (visão).

Fowler (2006) afirma que a separação entre a visão e o controlador não é tão importante, e recomenda-a só quando for útil. Na verdade, o termo correto é “quando for necessário”. Seu catálogo de padrões descreve três que funcionam como controladores e que podem atuar sozinhos ou combinados. A decisão de quais usar depende das responsabilidades do controlador.

Sommerville (2007, p. 245 e p. 283) acrescenta que a abordagem do MVC “é uma maneira eficiente de apoiar as várias apresentações

de dados”, pois permite que os dados sejam apresentados de maneiras diferentes com uma interação separada para cada uma dessas apresentações.

Bass et al. (2003) afirmam que interfaces do usuário são frequentemente revisadas durante o processo de teste de software. De acordo com os autores, isso leva o engenheiro de usabilidade a separar a interface do usuário do resto da aplicação, e ainda citam o MVC como um dos padrões de arquitetura que suportam a modificação da interface do usuário.

Modularização

Passos (2013) afirma, de forma aparentemente contraditória, que a solução para tratar os problemas de manutenção de software é “construir software a partir de blocos de construção complexos (componentes)”. Você pode pensar “não seria melhor construir a partir de blocos simples, pois eles seriam mais compreensíveis?”.

Segundo Heuser (2001, p. 5), “essa modularização de programas tem várias vantagens. A manutenção de programas torna-se mais simples, pois uma separação clara de funções torna programas mais facilmente compreensíveis. A produtividade de programadores também aumenta, já que os programas ficam menores, pois usam funções já construídas”.

Bloch (2006) afirma que “o bom código é modular”. Bloch, na verdade, defende o projeto de APIs, pois, segundo ele, “pensar em termos de APIs melhora a qualidade do código”. Uma API é um módulo pensado para servir a várias aplicações, concentrando lógica que é de domínio genérico. O autor enumera algumas características para uma boa API que podem, no entanto, ser aplicadas a outros tipos de estruturas modulares de código:

- Facilidade de aprendizado;
- Facilidade de uso, mesmo sem documentação;
- Dificuldade de fazer coisas erradas;

- Facilidade de ler e manter o código que a usa;
- Poder o suficiente para satisfazer aos requisitos;
- Facilidade de extensão;
- Apropriada para a audiência (desenvolvedores).

De acordo com Passos (2013), um componente pode conter grande complexidade.

Frameworks

Pressman (2004, p. 203) afirma que, para viabilizar o projeto de software baseado em padrão, “pode ser necessário fornecer uma infraestrutura do esqueleto de implementação específica, chamada de arcabouço (framework)”. Ele define framework como uma “miniarquitetura reusável, que fornece a estrutura e o comportamento genéricos para uma família de abstrações de software, dentro de um contexto (...) que especifica sua colaboração e uso em determinado domínio”.

Para Sommerville (2007, p. 283), um “framework é uma estrutura genérica que pode ser ampliada para criar um subsistema ou aplicação mais específica”. Para ele, “Ampliar um framework pode envolver a adição de classes concretas que herdam operações de classes abstratas no framework. Além disso, pode ser necessário definir callbacks que são chamadas em resposta a eventos reconhecidos pelo framework”.

Guerra e Yoder (2011) afirmam que “frameworks resolvem um conjunto particular de problemas que são adaptados para contextos específicos e podem prover um projeto otimizado para a aplicação”. Eles distinguem framework de bibliotecas de classes ou componentes, pois “componentes podem ser conectados na estrutura de um framework e são subclasses concretas que fazem o trabalho real”.

Conforme Gamma et al. (2000, p. 42), “o framework dita a arquitetura de sua aplicação. Ele vai definir a estrutura geral, sua

divisão em classes e objetos e em consequência as responsabilidades-chave das classes de objetos, como estas colaboram, e o fluxo de controle". Um framework, para eles, "predefine esses parâmetros de projeto, de maneira que o projetista ou implementador da aplicação possa se concentrar nos aspectos específicos da mesma".

Para Gamma et al., um framework captura as decisões de projeto que são comuns ao domínio da aplicação. Assim, frameworks enfatizam reutilização de projetos em relação à reutilização de código, embora um framework geralmente inclua subclasses concretas que o desenvolvedor possa usar diretamente. O reúso neste nível leva a uma inversão de controle entre a aplicação e o software sobre a qual ela está baseada.

Ainda segundo Gamma et al. (2000, p. 42), "se as aplicações são difíceis de projetar, e os toolkits são ainda mais difíceis, os frameworks, então, são os mais difíceis de todos". De acordo com eles, "o projetista de um framework aposta que uma arquitetura funcionará para todas as aplicações do domínio. Qualquer mudança substancial no projeto do framework reduziria seus benefícios consideravelmente, uma vez que a principal contribuição de um framework para uma aplicação é a arquitetura que ele define. Portanto é imperativo projetar o framework de maneira que ele seja tão flexível e extensível quanto possível".

Os autores citados anteriormente afirmam que os frameworks "são a maneira pela qual os sistemas orientados a objeto conseguem a maior reutilização". Segundo eles, "aplicações orientadas a objetos maiores terminarão por consistir-se de camadas de frameworks que cooperam uns com os outros. A maior parte do projeto e do código da aplicação virá dos frameworks que ela utiliza ou será influenciada por eles".

De acordo com Guerra e Yoder (2011), o reúso de código e o projeto proporcionados por um framework reduzem o tempo de desenvolvimento e o custo de manutenção.

Guerra e Yoder (2011) afirmam que a primeira regra para o projeto de um framework é: não construa um framework, em vez disso, compre um. Eles justificam sua recomendação declarando que é difícil encontrar os domínios genéricos que devem ser implementados pelo framework, e que o desafio do aprendizado e da construção é muito grande.

Desafios do programador

O processo de alienação do programador é inevitável, porque, segundo Djikstra (1972 apud MCCONNELL, 2005, p. 110), “ninguém tem um cérebro grande o bastante para conter um programa de computador moderno”. Se algo foge à nossa compreensão, também foge ao nosso domínio.

Passos (2013) afirma que “o trabalho do programador é desumano porque se exige que eles [sic] escrevam grandes quantidades de código complexo sem erros e em um curto espaço de tempo”.

Mitos da programação

De acordo com Martin (2009, p. 2), “algumas pessoas de fato sugerem que nós estamos próximos do fim do código. Que logo todo código será gerado em vez de ser escrito. Que os programadores não são necessários porque as pessoas de negócio vão gerar programas a partir de suas especificações”.

Martin (2009, p. 2) rebate essa ideia equivocada, afirmando que “nós nunca estaremos livres do código, porque o código representa os detalhes dos requisitos. Em alguns níveis esses detalhes não podem ser ignorados ou abstraídos; eles têm de ser especificados. E especificar requisitos em tal detalhe que uma máquina possa executá-los é programar. Tal especificação é código”.

Princípios da boa programação

Conforme Diggins (2011), os princípios da boa programação estão fortemente ligados aos princípios de bom projeto e engenharia. Ele relaciona alguns princípios que, por experiência própria, ao serem seguidos, possibilitam que qualquer desenvolvedor se torne mais eficiente e capaz de produzir código mais fácil de manter e com poucos defeitos.

Calistenia

Segundo Bay (2008), a calistenia é um tipo de ginástica criada em 1829 por Phoktion Heinrich Clias na França, e estabelecida nos Estados Unidos pelo doutor Dio Lewis e pela ACM (Associação Cristã de Moços). A proposta da calistenia era promover exercícios físicos de forma simples, cativante e fundamentada na ciência.

Como desenvolvedores de software adoram roubar ideias de outras áreas para aplicar em seu campo de trabalho, você não deve se surpreender ao saber que existe a calistenia de objetos. Seria algo mais ou menos assim: o programador faz os exercícios, mas são os objetos que entram em forma.

Programação orientada a objetos

De acordo com Shalloway (apud BAY, 2008), o bom projeto orientado a objeto tem sete qualidades: coesão, baixo acoplamento, nenhuma redundância, encapsulamento, testabilidade, legibilidade e foco.

Capítulo 13
Referências

BASS, L. et al. *Software Architecture in Practice*. 2. ed. Addison-Wesley, 2003.

BAY, J. Object Calisthenics: 9 steps to better software design today. In: *The ThoughtWorks Anthology:* Essays on Software Technology and Innovation. p. 70-80. The Pragmatic Bookshelf, 2008. Disponível em: http://www.bennadel.com/resources/uploads/2012/ObjectCalisthenics.pdf.

BLOCH, J. *How to Design a Good API and Why it Matters*. Proceedings OOPSLA '06 Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. p. 506-507. Out. 2006. Disponível em: https://dl.acm.org/citation.cfm?id=1176622.

BURNETTE, E.; HOUGHTON, A.; BUZIUK, I.; ISTRIA, M. *Getting Started with Eclipse*. Disponível em: http://refcardz.dzone.com/refcardz/getting-started-eclipse.

COPLIEN, J. O. *The DCI Architecture:* Lean and Agile at the Code Level. 2010. Disponível em: http://qconlondon.com/dl/qcon-london-2010/slides/JimO.Coplien_TheDCIArchitectureLeanAndAgileAtTheCodeLevel.pdf.

DEITEL, H. M.; DEITEL, P. J. *Internet & World Wide Web, como programar*. 2. ed. Porto Alegre: Bookman, 2003.

DIGGINS, C. *The Principles of Good Programming*. Jul. 2011. Disponível em: http://www.artima.com/weblogs/viewpost.jsp?thread=331531.

EELES, P. *What is a software architecture?* Fev. 2006. Disponível em: http://www.ibm.com/developerworks/rational/library/feb06/eeles/.

FOWLER, M. *Refatoração:* aperfeiçoando o projeto de código existente. Porto Alegre: Bookman, 2004.

FOWLER, M. *Padrões de desenvolvimento de arquiteturas corporativas*. Porto Alegre: Bookman, 2006.

GAMMA, E. et al. *Padrões de projeto:* soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

GILMORE, W. J. *PHP*. 2008. Disponível em: http://janetkulyk.com/pdf/php.pdf.

GLASS, R. L. *Software maintenance is a solution, not a problem*. Mar. 2006. Disponível em: http://www.developerdotstar.com/mag/articles/maintenance_solution.html.

GUERRA, E.; YODER, J. W. *An evolutionary vision of framework development*. Proceedings II Congresso Brasileiro de Software: Teoria e Prática. Set. 2011.

HATTORI, L. P. *Análise probalística de impacto de mudanças baseada em históricos de mudanças do software*. Tese de Mestrado, Universidade Federal de Campina Grande, Campina Grande, Brasil, fev. 2008. Disponível em: http://www.inf.usi.ch/phd/hattori/downloads/Dissertacao_LilePalmaHattori.pdf.

HELLERSTEIN, J. M.; STONEBREAKER, M.; HAMILTON, J. Architecture of a Database System. *Foundations and Trends in Databases,* v. 1, n. 2, p. 141-259, 2007. Disponível em: http://db.cs.berkeley.edu/papers/fntdb07-architecture.pdf.

HEUSER, C. A. *Projeto de banco de dados*. Porto Alegre: Sagra Luzatto, 2001.

HOGLUND, G.; MCGRAW, G. *Como quebrar códigos*: a arte de explorar (e proteger) software. São Paulo: Pearson Makron Books, 2006.

HOLT, B. *PHP 5.4:* scripting with PHP. Disponível em: http://refcardz.dzone.com/refcardz/php-54-scalable.

HOWARD, M.; LEBLANC, D. *Escrevendo código seguro*: estratégias e técnicas práticas para codificação segura de aplicativos em um mundo em rede. 2. ed. Porto Alegre: Bookman, 2005.

MARTIN, R. C. *Clean Code*: a handbook of Agile software craftsmanship. Boston: Pearson Education, 2009.

MCCONNELL, S. *Code Complete*: um guia prático para a construção de software. 2. ed. Porto Alegre: Bookman, 2005.

MCCULLOUGH, M. *Getting Started with GIT*. Disponível em: http://people.inf.ethz.ch/mkovatsc/wiki/_media/general:other:rc094-010d-git_1.pdf.

MELIA. C. *The MVVM Design Pattern*. 2013. Disponível em: http://refcardz.dzone.com/refcardz/mvvm-design-pattern-formula.

METSKER, S. J. *Padrões de projeto em Java*. Porto Alegre: Bookman, 2004.

IDE, A. *PHP just grows & grows*. Jan. 2013. Disponível em: http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html. Acesso em 14/01/2017.

PAGE-JONES, M. *O que todo programador deveria saber sobre projeto orientado a objeto*. São Paulo: Makron Books, 1997.

PARISH, G. *Técnicas de manutenção de programas e sistemas*. Rio de Janeiro: LTC, 1990.

PASSOS, J. *Engenharia de software: o futuro do software*. 2013. Disponível em:

http://www.facape.br/jocelio/es/material_em_pdf_conceitual/ESFuturoSW.pdf.

PATRICK, J. J. *SQL fundamentos*. 2. ed. São Paulo: Berkeley, 2002.

PAULA FILHO, W. P. *Engenharia de software*: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009.

PHP-FIG. *FIG Standards*. Disponível em: https://github.com/php-fig/fig-standards. Acesso em: 17 jan. 2013.

POTENCIER, F. *Dependency Injection with PHP 5.3... with a bit of PHP 5.4*. Out. 2011. Disponível em: http://www.slideshare.net/fabpot/dependency-injection-in-php-5354.

PRESSMAN, R. S. *Engenharia de software* 4. ed. São Paulo: Mcgraw-Hill, 2004.

SILVA, R. P.; PRICE, R. *A busca de generalidade, flexibilidade e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos*. 1998. Disponível em: http://www.inf.ufsc.br/~ricardo/publications/Ideas98.PDF.

REENSKAUG, T.; COPLIEN, J. O. *The DCI architecture*: a new vision of object-oriented programming. Mar. 2009. Disponível em: http://www.artima.com/articles/dci_vision.html.

RIBEIRO, F. *PHP Jedi:* boas práticas e alta performance. Mar. 2009. Disponível em: http://www.slideshare.net/felipernb/php-jedi-boas-prticas-e-alta-performance.

SHAFIK, D.; RAMSEY, B. *PHP|architect's Zend PHP 5 Certification Guide*. Toronto: Marco Tabini & Associates, 2006.

SCHINDLER, R. *Learning about dependency injection and PHP*. Maio. 2011. Disponível em: http://ralphschindler.com/2011/05/18/learning-about-dependency-injection-and-php.

SIRONI, G. *PHPUnit*: PHP Test-Driven Development. Disponível em: http://refcardz.dzone.com/refcardz/phpunit-php-test-driven.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

SWEAT, J. E. *php|architect's Guide to PHP Design Patterns*. Toronto: Marco Tabini & Associates, 2005.

TAKAHASHI, M. *Guia mangá de bancos de dados*. São Paulo: Novatec, 2009.

TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. *Estruturas de dados usando C*. São Paulo: Pearson Education do Brasil, 1995.

TKOTZ, V. *Criptografia*: segredos embalados para viagem. São Paulo: Novatec, 2005.

VENNERS, B. *A conversation with Erich Gamma*. Maio. 2005. Disponível em: http://www.artima.com/lejava/articles/gammadp.html.

W3TECHS. *Usage statistics and market share of PHP for websites*. Disponível em: http://w3techs.com/technologies/details/pl-php/all/all. Acesso em: 14 jan. 2017.

WILTON, P.; COLBY, J. W. *Beginning SQL*. 2. ed. Indianapolis: Wiley Publishing, 2005.